

**DYNAMIC ROUND-ROBIN PEER-TO-PEER (P2P)**

**DOMAIN NAME SYSTEM (DNS)**

BY

**FAHD AHMED ABDULRAHMAN ABDULHAMEED**

A Thesis Presented to the  
DEANSHIP OF GRADUATE STUDIES

**KING FAHD UNIVERSITY OF PETROLEUM & MINERALS**

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the  
Requirements for the Degree of

**MASTER OF SCIENCE**

In

**COMPUTER NETWORKS**

**NOVEMBER 2010**

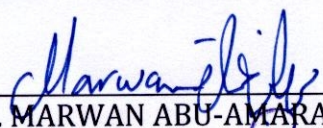
# KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

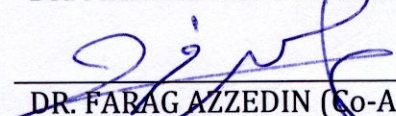
DHAHRAN 31261, SAUDI ARABIA


## DEAN OF GRADUATE STUDIES

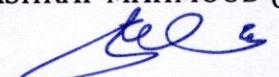
This thesis, written by FAHD AHMED ABDULRAHMAN ABDULHAMEED under the direction of his thesis advisor and approved by his thesis committee, has been presented to and accepted by the DEAN OF GRADUATE STUDIES, in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE IN COMPUTER NETWORKS.

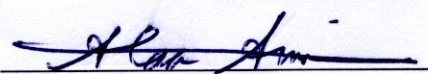
### Thesis Committee

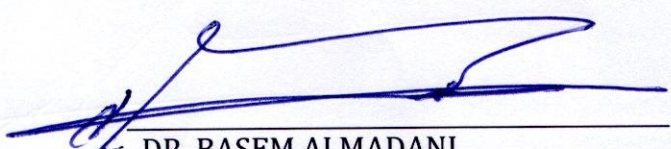
  
DR. MARWAN ABU-AMARA (Advisor)


  
DR. FARAG AZZEDIN (Co-Advisor)

  
DR. ASHRAF MAHMOUD (Member)

  
DR. MOHAMMAD SQALLI (Member)

  
DR. ALAAELDIN AMIN (Member)

  
DR. BASEM ALMADANI  
(Chairman of Computer Engineering)

  
DR. SALAM A. ZUMMO  
(Dean of Graduate Studies)

Date

1/2/11





# **DEDICATION**

To My Family

To Dr. Marwan Abu-Amara

## **ACKNOWLEDGEMENTS**

All praises to the Almighty ALLAH, the most Gracious, the most Merciful, who gave me the courage and patience to carry out this work. Then, I pray to ALLAH to bestow peace on his Prophet Muhammad, peace be upon him.

Acknowledgement is due to King Fahd University of Petroleum & Minerals (KFUPM) for supporting this research, and the funding provided by King Abdul-Aziz City for Science and Technology (KACST) under the National Science and Technology Plan (project number 08-INF97-4).

I would like to deeply thank my major advisor Dr. Marwan Abu-Amara for his trust, guidance and support throughout this thesis. I also thank my thesis co-advisor Dr. Farag Azzedin for his guidance and sharing ideas of this work. I wish to thank Dr. Ashraf Mahmoud, Dr. Mohammad Sqalli, and Dr. Alaaeldin Amin, for serving as my committee members, and giving many insightful suggestions which improved this thesis.

Finally, I would like to express special thanks to my family for their support.

# TABLE OF CONTENTS

DEDICATION.....	III
ACKNOWLEDGEMENTS.....	IV
TABLE OF CONTENTS.....	V
LIST OF TABLES.....	X
LIST OF FIGURES.....	XI
THESIS ABSTRACT .....	XV
ملخص الرسالة.....	XVI
CHAPTER 1 INTRODUCTION.....	1
1.1. THESIS CONTRIBUTIONS .....	2
1.2. THESIS ORGANIZATION .....	3
CHAPTER 2 BACKGROUND .....	4
2.1. DOMAIN NAME SYSTEM .....	4
2.1.1. Introduction.....	4
2.1.2. DNS Lookup Process.....	5

2.2. CHORD PROTOCOL .....	7
2.2.1. Chord Routing Information.....	8
2.2.2. Lookup Process.....	9
2.2.3. Stabilization Process.....	11
2.2.4. Joining Process.....	13
2.2.5. Leaving Process.....	14
<b>CHAPTER 3 LITERATURE REVIEW .....</b>	<b>15</b>
3.1. DNS THREATS.....	15
3.2. DNS COUNTERMEASURES FOR MALICIOUS ATTACKS .....	17
3.3. PEER-TO-PEER SOLUTIONS FOR DNS .....	18
3.4. LOAD BALANCING IN PEER-TO-PEER.....	19
<b>CHAPTER 4 INTENTIONAL DENIAL OF SERVICE TO DNS.....</b>	<b>21</b>
4.1. PROBLEM DESCRIPTION.....	21
4.2. SOLUTION OBJECTIVES .....	22
<b>CHAPTER 5 DYNAMIC ROUND-ROBIN P2P .....</b>	<b>24</b>
5.1. INTRODUCTION.....	24
5.2. ASSUMPTIONS.....	25
5.3. NODE STATES .....	26
5.4. PROPOSED SOLUTION .....	29

5.5. METHODOLOGY .....	31
5.6. MODIFIED CHORD PROTOCOL .....	32
5.6.1. <i>Find Successor Procedure</i> .....	32
5.6.2. <i>Closest Predecessor Procedure</i> .....	34
5.6.3. <i>Notify Neighbors</i> .....	35
5.7. MODIFIED ROUND-ROBIN .....	35
5.7.1. <i>Lookup Window</i> .....	36
5.7.2. <i>Lookup Procedure</i> .....	36
5.8. COMPLEXITY AND LIMITATIONS .....	38
<b>CHAPTER 6 SIMULATOR .....</b>	<b>39</b>
6.1. SIMULATOR MODULES .....	40
6.1.1. <i>Core Modules</i> .....	40
6.1.2. <i>Simulation Modules</i> .....	41
6.1.3. <i>Analysis Modules</i> .....	44
6.1.4. <i>Miscellaneous Modules</i> .....	46
6.2. SIMULATOR'S ASSUMPTIONS .....	46
6.3. SIMULATOR PARAMETERS .....	47
6.3.1. <i>General Parameters</i> .....	47
6.3.2. <i>Random Variables</i> .....	49

6.3.3. Network Parameters.....	50
6.3.4. Node Parameters.....	50
6.4. PERFORMANCE METRICS .....	51
6.5. VERIFICATION.....	54
6.5.1. Scenario 1: Load Balance Scenario.....	54
6.5.2. Scenario 2: Path Length Scenario .....	56
6.5.3. Scenario 3: Nodes Failure Scenario.....	57
6.5.4. Scenario 4: Joining and Leaving Scenario .....	58
<b>CHAPTER 7 RESULTS AND ANALYSIS .....</b>	<b>61</b>
7.1. SCENARIO PARAMETERS .....	62
7.2. BLOCKING SCENARIO.....	63
7.2.1. Objective.....	63
7.2.2. Scenario Description.....	63
7.2.3. Scenario Results .....	64
7.2.4. Observations.....	73
7.3. FAILURE SCENARIO.....	74
7.3.1. Objective.....	74
7.3.2. Scenario Description.....	74
7.3.3. Scenario Results .....	74
7.3.4. Observations.....	83



7.4. JOINING AND LEAVING SCENARIO .....	84
7.4.1. <i>Objective</i> .....	84
7.4.2. <i>Scenario Description</i> .....	85
7.4.3. <i>Scenario Results</i> .....	85
7.4.4. <i>Observations</i> .....	92
7.5. EVALUATION AND RECOMMENDATIONS .....	93
<b>CHAPTER 8 CONCLUSION</b> .....	<b>97</b>
8.1. FUTURE WORK.....	98
<b>REFERENCES</b> .....	<b>99</b>
<b>VITA</b> .....	<b>105</b>

## LIST OF TABLES

Table 5-1: The difference between Dynamic Round-robin P2P and Chord protocol.....	25
Table 6-1: Simualtor's counters descriptions .....	45
Table 6-2: Simualtor's miscellaneous modules .....	46
Table 6-3: Default values for simualtor's general parameters .....	48
Table 6-4: Simualtor's random variable descriptions and default seeds.....	49
Table 7-1: Scenarios' default parameters .....	62

## LIST OF FIGURES

Figure 2.1: The DNS operation.....	6
Figure 2.2: Chord lookup process.....	10
Figure 2.3: Chord lookup example .....	11
Figure 2.4: Chord stabilization process .....	12
Figure 2.5: Chord joining procedure .....	13
Figure 2.6: Chord leaving procedure .....	14
Figure 4.1: An illustration of the blocking behavior at KSA.....	22
Figure 5.1: Node states.....	29
Figure 5.2: An illustration of the proposed solution at KSA .....	31
Figure 5.3: Modified Find Successor procedure.....	33
Figure 5.4: Modified Closest Predecessor procedure .....	34
Figure 5.5: Lookup process in the modified round-robin.....	37
Figure 6.1: Core modules in the simulator .....	41
Figure 6.2: Simulation process .....	44
Figure 6.3: Load Balance Scenario .....	55

Figure 6.4: Load Balance Scenario with different virtual nodes.....	55
Figure 6.5: Path Length Scenario.....	56
Figure 6.6: Nodes Failure Scenario – Timeout.....	57
Figure 6.7: Nodes Failure Scenario - Path length.....	58
Figure 6.8: Joining and Leaving Scenario - Timeout.....	59
Figure 6.9: Joining and Leaving Scenario - Path length.....	59
Figure 6.10: Joining and Leaving Scenario - Lookup failure .....	60
Figure 7.1: Blocking Scenario - Lookup failure percentage with 10,000 queries.....	65
Figure 7.2: Blocking Scenario - Lookup failure percentage with 100,000 queries.....	65
Figure 7.3: Blocking Scenario - Average load between the resolvers.....	66
Figure 7.4: Blocking Scenario – The average, 1 <sup>st</sup> and 99 <sup>th</sup> percentiles, load between the resolvers in Round-10 and Chord-20 with four virtual nodes.....	67
Figure 7.5: Blocking Scenario - Load fairness between the resolvers with 10,000 queries .....	68
Figure 7.6: Blocking Scenario - Load fairness between the resolvers with 100,000 queries .....	68
Figure 7.7: Blocking Scenario - Successful lookup path length .....	69
Figure 7.8: Blocking Scenario - Successful lookup timeout.....	70

Figure 7.9: Blocking Scenario - Average traffic between the nodes .....	71
Figure 7.10: Blocking Scenario - Traffic fairness between the nodes with 10,000 queries .....	72
Figure 7.11: Blocking Scenario - Traffic fairness between the nodes with 100,000 queries .....	72
Figure 7.12: Failing Scenario - Lookup failure percentage with 10,000 queries.....	75
Figure 7.13: Failing Scenario - Lookup failure percentage with 100,000 queries.....	76
Figure 7.14: Failing Scenario - Average load between the resolvers.....	77
Figure 7.15: Failing Scenario - The average, 1 <sup>st</sup> and 99 <sup>th</sup> percentiles, load between the resolvers in Round-10 and Chord-20 with four virtual nodes.....	78
Figure 7.16: Failing Scenario - Load fairness between the resolvers .....	78
Figure 7.17: Failing Scenario - Successful lookup path length with 10,000 queries.....	79
Figure 7.18: Failing Scenario - Successful lookup path length with 100,000 queries.....	80
Figure 7.19: Failing Scenario - Successful lookup timeout with 10,000 queries .....	81
Figure 7.20: Failing Scenario - Successful lookup timeout with 100,000 queries.....	81
Figure 7.21: Failing Scenario - Average traffic between the nodes .....	82
Figure 7.22: Failing Scenario - Traffic fairness between the nodes.....	83

Figure 7.23: Joining & Leaving Scenario - Lookup failure percentage .....	86
Figure 7.24: Joining & Leaving Scenario - Average load between the resolvers.....	87
Figure 7.25: Joining & Leaving Scenario – The average, 1st and 99th percentiles, load between the resolvers in Round-10 and Chord-20 with four virtual nodes .....	88
Figure 7.26: Joining & Leaving Scenario - Load fairness between the resolvers.....	88
Figure 7.27: Joining & Leaving Scenario - Successful lookup path length .....	89
Figure 7.28: Joining & Leaving Scenario - Successful lookup timeout.....	90
Figure 7.29: Joining & Leaving Scenario - Average traffic between the nodes .....	91
Figure 7.30: Joining & Leaving Scenario - Traffic fairness between the nodes .....	92
Figure 7.31: Comparison between Round-10 and Chord-20: Load .....	94
Figure 7.32: Comparison between Round-10 and Chord-20: Load Fairness .....	94
Figure 7.33: Comparison between Round-10 and Chord-20: Lookup Failure.....	95
Figure 7.34: Comparison between Round-10 and Chord-20: Path lenght .....	95
Figure 7.35: Comparison between Round-10 and Chord-20: Timeout.....	95
Figure 7.36: Comparison between Round-10 and Chord-20: Traffic .....	96
Figure 7.37: Comparison between Round-10 and Chord-20: Traffic Fairness.....	96

# THESIS ABSTRACT

**NAME:** FAHD AHMED ABDULRAHMAN ABDULHAMEED  
**TITLE:** DYNAMIC ROUND-ROBIN PEER-TO-PEER (P2P) DOMAIN NAME SYSTEM (DNS)  
**MAJOR FIELD:** COMPUTER NETWORKS  
**GRADUATION DATE:** 8<sup>th</sup> November 2010

The Domain Name System (DNS) is a critical service in the Internet infrastructure, and it provides user-friendly name to Internet IP address mapping services. The absence of the DNS has a severe impact on several Internet applications such as HTTP, FTP, and/or email that can cause such applications to become non-functional. To avoid an intentional blocking from higher name servers, a dynamic round-robin P2P solution that is built over Chord protocol is proposed as a secondary path to resolve the DNS queries. An evaluation of the proposed solution based on load balancing, failure, timeout, and number of hops is also conducted. Furthermore, the effect of joining and leaving nodes is also considered in the study.



## ملخص الرسالة

الاسم:

فهد أحمد عبدالرحمن محمد عبدالحמיד

عنوان الرسالة:

نظام أسماء النطاقات الديناميكية باستخدام تقنية السلسلة المتعاقبة (Round-Robin) و بروتوكول الند للند (P2P)

التخصص:

شبكات الحاسب الآلي

تاريخ التخرج:

٨ نوفمبر ٢٠١٠

نظام أسماء النطاقات (DNS) هي خدمة حاسمة في البنية التحتية للإنترنت التي توفر خدمة تحويل الروابط النصية إلى عناوين رقمية (IP). وعدم وجود نظام أسماء النطاقات له تأثير شديد على العديد من تطبيقات الإنترنت ، مثل بروتوكول تصفح المواقع ، بروتوكول نقل الملفات ، و / أو البريد الإلكتروني، والتي يمكن أن تسبب إلى توقف وظيفة هذه البروتوكولات. فلتجنب الإساءة المتعمدة من الملقمات الأعلى في نظام أسماء النطاقات، فإن هذا البحث يقدم حل مرناً يعتمد على تقنية السلسلة المتعاقبة (Round-Robin) و بروتوكول الند للند (P2P) ويعتبر الحل كمسار ثاني للرد على استفسارات نظام أسماء النطاقات. ولقد تم أيضاً تقييم الحل المقترح على أساس الموازنة، والفشل في الرد على الاستفسارات، وفترات الانتظار، وعدد القفزات. كما أنه تم دراسة تأثير انضمام أعضاء جدد في بروتوكول الند للند أو مغادرة أعضاء فاعلين على الحل المقترح.

# CHAPTER 1

## INTRODUCTION

The Internet was started as a small project that is called ARPANET in 1958, and it continued to grow exponentially serving 1.97 billion users as of 30 June 2010 [1]. Nowadays, the Internet plays an important role in modern societies where communications, research, and businesses have become more dependent on the Internet.

One of the critical services in the Internet infrastructure is the Domain Name System (DNS) that provides user-friendly name to Internet IP address mapping services. The DNS consists of a hierarchy of DNS servers with 13 root nameservers at the top of the hierarchy. The IP addresses of the 13 root nameservers are hard coded in **root hints** file in every recursive or caching DNS server [2].

The absence of the DNS has a severe impact on several Internet applications such as HTTP, FTP, and/or email that can cause such applications to become non-functional. Denial of service, cache poisoning, and compromised data are examples of DNS malicious threats. In the literature, many papers discussed the effects of Denial of Service (DoS) attacks on the DNS servers. However, to the best of our knowledge, no research was conducted on the intentional denial of service to the

DNS service (i.e., root nameservers) by a service provider. This thesis discusses the problem of intentional denial of service to the DNS service by a service provider, and proposes a peer-to-peer (P2P) solution. The proposed P2P solution is based on the Chord protocol [3].

## **1.1. Thesis Contributions**

1. Study the performance of the Chord protocol that proposed by Stoica et al. taking into account the worst case conditions.
2. Develop and implement a new robust solution to overcome intentional DNS blocking from higher DNS servers without any modification to the existing DNS standard.
3. Implement an object oriented simulator to simulate a modified Chord protocol and the proposed solution.
4. Study the performance of the proposed solution from different aspects taking into account the worst case conditions.
5. Compare the performance of the proposed solution with the performance of the Chord protocol, and, accordingly, provide recommendations.

## **1.2. Thesis Organization**

CHAPTER 2 provides a brief summary about the existing DNS and the Chord protocols. It is followed by a literature review chapter that states different DNS threats and possible countermeasures with focus on P2P solutions. Problem description and solution approach are described in the fourth and fifth chapters, respectively. The sixth chapter discusses in details our developed simulator that we used to measure the performance of our proposed solution. Simulation results and analysis are discussed in the seventh chapter. Finally, a conclusion of the thesis and possible future work are given in the last chapter.

## CHAPTER 2

### BACKGROUND

In this chapter, a simple background of the Domain Name System (DNS) and the Chord protocol is presented.

#### 2.1. Domain Name System

##### 2.1.1. Introduction

The DNS could be defined as a hierarchical system of distributed **name servers**. Each name server holds a subset of the **DNS space in the form of Resource Records (RRs)** that belong to its zone as well as cached RRs. These RRs are used to resolve the queries that are being sent by clients through **resolvers** [4]. On the top of the DNS hierarchy are the root nameservers that are considered a critical part of the Internet because they are the first step in resolving the Resolvers' queries [5]. For this reason, there are 13 distributed nameservers and their IPs are hard coded in the **root hints** file in every recursive or caching DNS server [2]. The second important type of name servers is the Top Level Domain

(TLD) which is divided into two categories: the generic TLDs (gTLDs) and the country code TLDs (ccTLDs). From TLDs, the rest of the DNS servers in the tree-structure extends [6]. Finally, the DNS server that is responsible for adding, removing, or updating the RRs that belong to its zone is called an authoritative name server.

### 2.1.2. DNS Lookup Process

The purpose of the DNS is to map the host name to the host's IP address. The DNS is also used to retrieve the RRs of the authoritative name server, the aliases of the host name, mail exchange records, different pointers, and transfer the entire zone [7]. The descriptions of the different types of the RRs are found in RFC1034 [4] and RFC1035 [8]. In general, when the resolvers (or nameserver) receive a request from the client, it proceeds according to the following process:

1. Check the local cache. If found, send the answer to the client; otherwise go to step 2.
2. Starting from the highest domain name server cached (i.e., root nameservers), lookup for the child domain. This process continues downward until the query is resolved otherwise an error is sent to the client.

A simple illustration for the DNS lookup process without caching is shown in Figure 2.1. When a user browses the Internet and tries to reach a specific URL, the DNS client in his local computer will take care of getting the IP address of the hosting server of this URL. The process starts from the local computer by sending a request to the root server asking about the IP address of the primary nameserver (TLD). After receiving the root server's response, a new request is sent to the primary nameserver asking about the IP address for the second nameserver; if any. This step is repeated until it reaches the last nameserver included in the URL. Afterwards, the connection with the hosting server could be established after receiving a response of the last nameserver regarding the IP address of the hosting server.

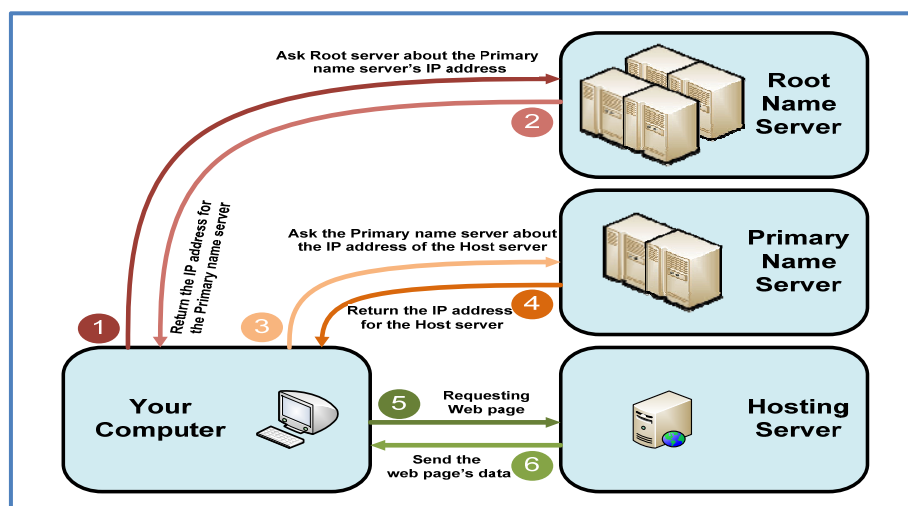


Figure 2.1: The DNS operation



## 2.2. Chord Protocol

A peer-to-peer (P2P) network is an overlay network that connects a group of peers with each other to share their resources (CPU, storage, content). Each peer is considered as a client and a server at the same time. This architecture provides a scalable and reliable network with the cost spread between the peers. The P2P networks can be classified either as an unstructured or as a structured P2P overlay network [9]. The unstructured P2P network consists of peers that communicate with each other through flooding without knowledge of the topology. The famous unstructured P2P protocols are Freenet [10], Gnutella [11], FastTrack [12], KaZaA [13], BitTorrent [14], Overnet [15], and eDonkey2000 [16]. On the other hand, each peer in the structured P2P network is associated with a subset of objects and the lookup is done through an efficient routing that is constructed by distributed routing structures, known as Distributed Hash Table (DHT). The famous structured P2P protocols are Content Addressable Network (CAN) [17], Tapestry [18], Chord [3][19], Pastry [20], Kademlia [21], and Viceroy [22].

Chord is a structured P2P overlay protocol with a simple mechanism to build a routing table that guarantees the success of the lookup in  $O(\log_2 N)$  hops where  $N$  is the network size[9]. Chord could be considered as a flat ring with nodes distributed clockwise in the ring in the order of their identifiers (IDs) that are generated by hashing the nodes' IP addresses. The outcomes of hashing all possible

lookup objects are referred to as the Chord space. The size of the Chord space is  $2^m$ , where  $m$  is the node ID length in binary. In the Chord protocol, each node is responsible for a subset of keys of the Chord space. This subset of keys consists of all the keys after the previous node in the ring to the current node included [3]. Some of the advantages of the Chord protocol are self-organization (i.e., no central management), scalability, robustness during network activity (i.e., joining and leaving), and successful routing even if the routing information is partially correct[9][3].

### 2.2.1. Chord Routing Information

Chord depends on three routing information that periodically get updated through a stabilization process that is explained in section 2.2.3 to resolve the incoming queries [3]. The three routing information are summarized in the following list:

1. **Predecessor:** a single record maps to the previous node in the ring.
2. **Successor List:** a minimum of one record maps to the next node in the ring.

There could be  $r$  subsequent nodes that provide better lookup and high stability when there are failures in the network.

3. **Finger Table:** selective nodes from the network that minimize the number of hops to find the successor. The finger table consists of  $m$  records (fingers) that satisfy the following function:

$$finger_{ID}(k) = successor((ID + 2^{k-1}) \bmod 2^m) \quad \text{where } 1 \leq k \leq m$$

The first finger in the finger table is the same as the first successor. Even though the finger table consists of  $m$  fingers, there are actually  $O(\log_2 N)$  unique fingers.

### 2.2.2. Lookup Process

The node is considered to have resolved the key that is a result of hashing the query when it finds the responsible node for this key. As shown in Figure 2.2, there are two routines used to resolve the query (i.e., key); Find Successor routine and Closest Predecessor routine. The Find Successor routine is used to find the successor to the key among the local successor list, and if there was no match it goes to the next routine. In the closest predecessor routine, the current node tries to forward the query to the nearest predecessor to the responsible node. The same process is used in forwarding the query until the query reaches the responsible node [3].

An example of a query lookup in Chord is shown in Figure 2.3. In this example, the successor list consists of only one record. Suppose a query for key-4 is submitted to node-10. Node-10 does not find the successor for key-4 in its local successor list, but it finds that node-3 is the nearest predecessor to key-4. After node-3 receives the forwarded query, it finds that node-5 is the successor for key-4 and the query is resolved by node-5.

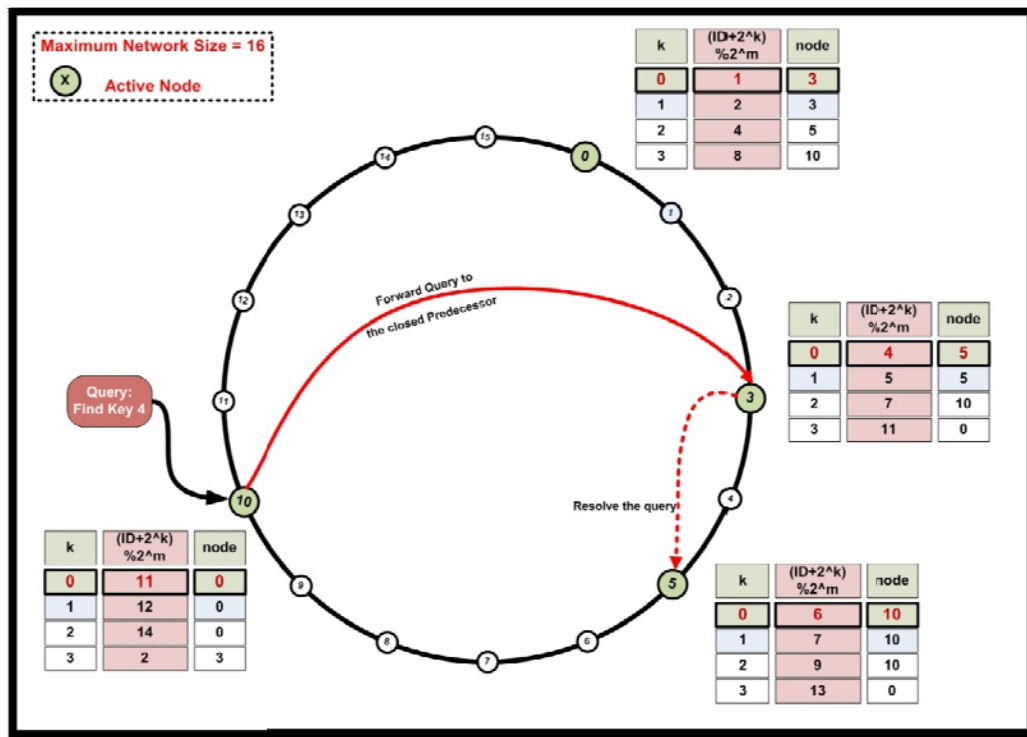


Figure 2.3: Chord lookup example

### 2.2.3. Stabilization Process

The stabilization process is used in Chord protocol to periodically update the successor list and one finger from the finger table. It takes  $m$  rounds to completely update the finger table. The stabilization process consists of three routines; Check Predecessor routine, Fix Successor routine, and Fix Finger routine as shown in Figure 2.4. The Check Predecessor routine is used to make sure that the previous peer in the ring is still alive; otherwise it will remove it from the

routing information to make space for a new predecessor. On the other hand, the Fix Successor routine is the main routine that makes sure that the successor list is up to date. It works by copying the successor list of the first live node in the local successor list. If there is no live node in the local successor list, it will use the lookup process described in subsection 2.2.2 to find the successor along with its successor list. The last routine, Fix Finger, is used to update the finger table that is done by checking one finger entry every round using the equation below:

$$finger_{ID}(k) = successor((ID + 2^{k-1}) \bmod 2^m) \quad \text{where } 1 \leq k \leq m$$

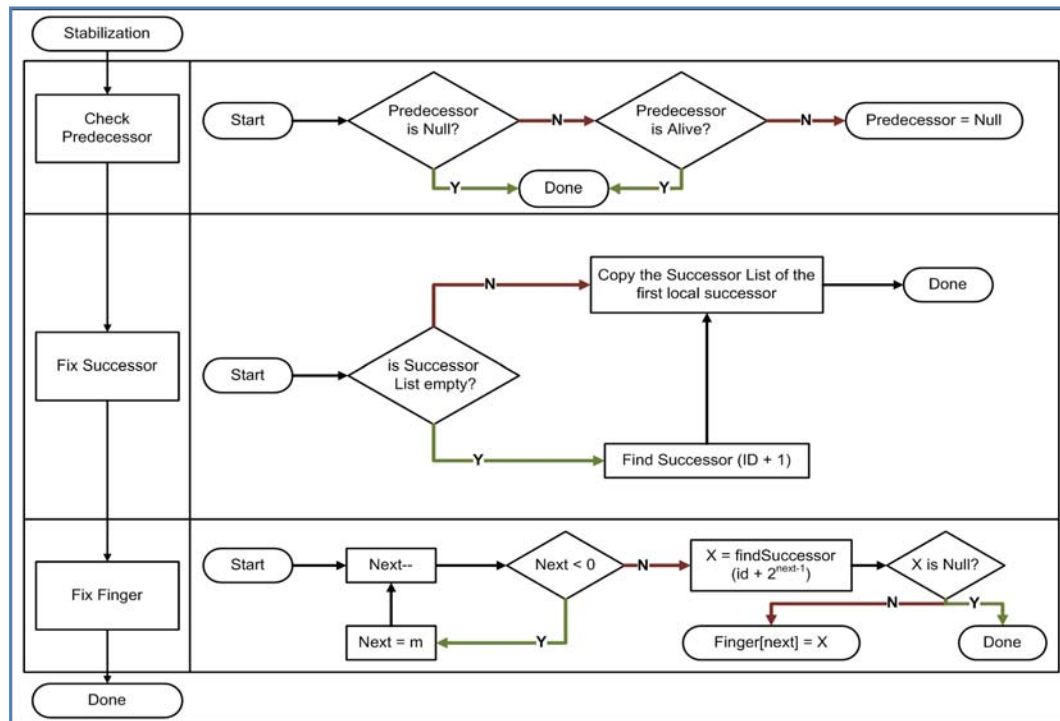


Figure 2.4: Chord stabilization process

### 2.2.4. Joining Process

By knowing a single live node, say node A, in the Chord ring, a new node could easily join the Chord network by asking node A to find the successor for the new node ID. After finding the successor, say node A', the successor list of the joining node will be built by copying the successor list of node A' and the routing information will be updated with each successive stabilization process execution [3]. As illustrated in Figure 2.5, node-4 asks node-10 about key-4's successor that is node-5. After that, the routing table will be copied from node-5 to node-4 and updated during the stabilization process.

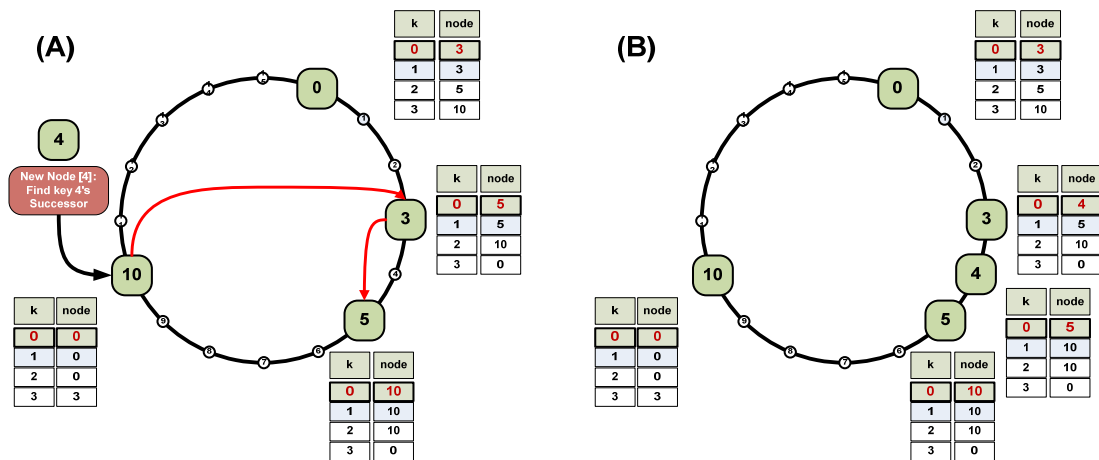


Figure 2.5: Chord joining procedure



### 2.2.5. Leaving Process

When a node voluntarily departs from the network, it will send a notification to its direct successor and predecessor that it is leaving along with information that will help them update their routing information with the correct successor and predecessor information [3]. Node-5 in Figure 2.6 sends messages to its predecessor and successor before leaving the network. Node-4 and node-10 update their routing table according to these messages and after several stabilization processes, node-5 will not appear in any routing table in the network.

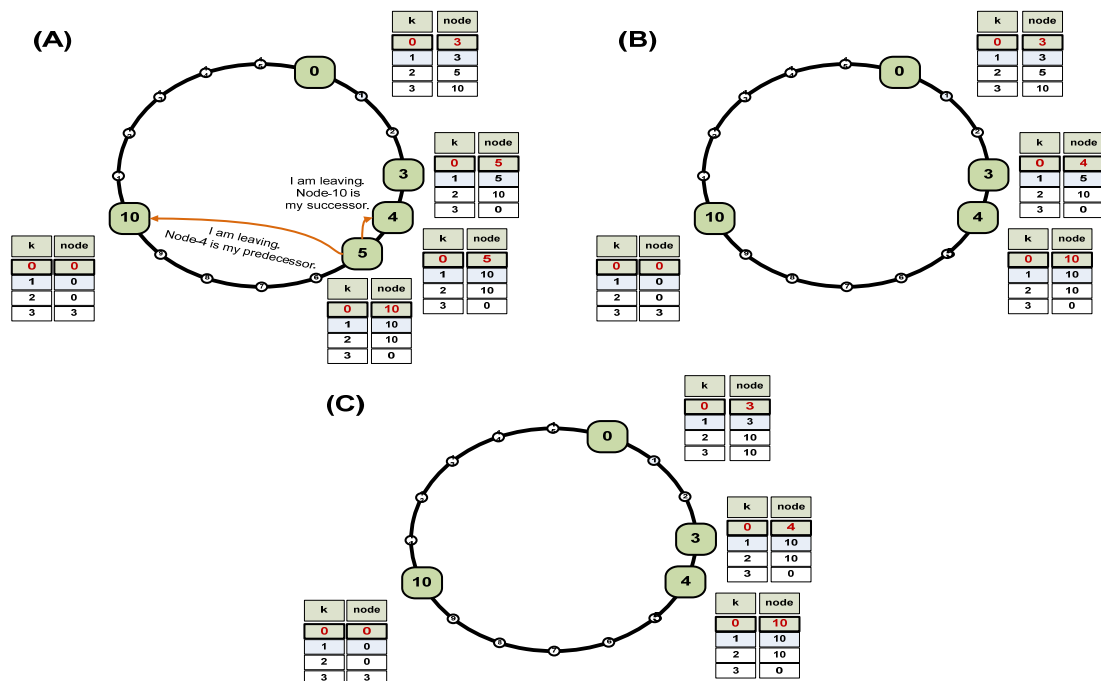


Figure 2.6: Chord leaving procedure

## CHAPTER 3

### LITERATURE REVIEW

In this chapter, we list some of the activities that can impact the DNS operation with emphasis on the malicious activities as they relate to the problem considered in the thesis. Also, we provide a review of some proposed P2P solutions that address the malicious activities targeting the DNS. Further, we consider the work done regarding the load balancing aspect of the proposed P2P solutions.

#### 3.1. DNS Threats

The DNS services could be impacted by both non-malicious and malicious activities. For example, the DNS service could be interrupted because of non-malicious activities such as erroneous operation or misconfiguration of the DNS servers that could cause unavailability of the DNS servers [23].

On the other hand, malicious activities, including denial of service (DoS), are another reason for interrupting the DNS service. In the past, several DoS attacks targeted the DNS system at different levels. For example, in October 2002, a

distributed denial of service (DDoS) attack on root nameservers ended up with seven of the root nameservers being completely down and two other nameservers being severely affected[24]. Another DDoS attack happened on February 2007 but the damages were less severe than the previous DDoS, because of the countermeasures of "Anycast" addressing and distributed servers that were put in place after the first attack [25]. Using these techniques helped in distributing the load of the root server to different geographical distributed servers that are working as a single root server.

Cheung [26] described how the DNS system can be attacked. Cheung divided the DNS system into 3 sections; the low level servers (resolvers), the communications between the servers and the clients, and the top level servers (nameservers). In the resolvers, the attackers try to get access to the DNS system through the available vulnerabilities which could lead to damaging the resolver servers or misbehavior of the resolver servers. By flooding the DNS servers with a massive number of forged queries, the normal DNS queries could be dropped in the communication line (i.e., routers). This could also happen if the attacker was able to take advantage of a critical router along the DNS server path to cause it to misbehave. Finally, the nameservers could be damaged by directly accessing the nameservers or through damaging other necessary services to the nameservers. Other malicious attacks that take advantage of the caching aspect in the DNS system include cache poisoning. In this type of attack, the attacker inserts an

incorrect DNS record or modifies an available DNS record in the DNS cache server so that it redirects the client to the attacker's website [27][28].

### **3.2. DNS Countermeasures for Malicious Attacks**

To countermeasure malicious attacks on the DNS several solutions were proposed. As stated in the previous section, the robustness of the name servers could be increased by using "Anycast routing" that counters the flooding attacks [26]. Another countermeasure is using the DNS security extensions (i.e., DNSSEC) that provide a secure integrity and authenticity to the DNS traffic [29]. Ballani and Francis [30] and Pappas et al. [24] discussed how to manipulate the Time-to-Live (TTL) value of the DNS records to improve the availability of the DNS system against the DoS attack. Edith and Haim[31] provided another solution that increases the efficiency of the DNS caching by suggesting two policies. The first policy is renewal cache refreshment where the DNS record will be refreshed upon the expiration of the TTL. The second policy is simultaneous validation where the expired DNS record will be refreshed whenever a DNS lookup requests this record. The peer-to-peer solutions are provided in the following section.

### 3.3. Peer-to-Peer Solutions for DNS

Several papers proposed a structured P2P solution either as a new DNS design or as an alternative DNS lookup service. The structured P2P system associates nodes (i.e., peers) with objects (i.e., keys) and constructs distributed routing structures, known as Distributed Hash Table (DHT), to support efficient routing between peers.

To ensure the availability of the root DNS servers, a secondary DNS lookup service through a cooperative cache sharing network that is based on Pastry and Chord protocols was proposed by Poolsappasit and Ray [27]. They concluded that by distributing the resource records of the domain in different cache resolvers, the effort to attack this domain becomes harder. Meanwhile, by using the cooperative network, it will only take  $O(\log_2 N)$  to find the record.

A new platform was proposed by Ramasubramanian and Sirer [32] that implements a Cooperative Domain Name System (CoDoNS) based on a structured P2P Pastry protocol with analytically-informed proactive caching (Beehive). It uses Beehive to distribute the popular records in the nodes near the home node (i.e., responsible node) and during the lookup it checks the cached records in the path that results in low lookup delay compared to the existing DNS. An alternative solution, proposed by Cox et al. [33] uses Chord protocol instead of the existing DNS. They called the prototype DDNS. In addition to implementing all the

functionality of the standard DNS, the new domain record in the DDNS requires authentication (i.e., digital signature) from the higher level. Although their design had more availability and better fault-tolerance, it suffered from high latency.

### **3.4. Load Balancing in Peer-to-Peer**

In P2P solutions, the load balancing is a critical issue that is discussed in different papers. The load could be objects' popularity, peers' homogeneousness, peers' activities, peers' locality (i.e., proximity-aware), or load types (i.e., stored objects, lookup traffic, responsible subset of the objects space).

Karger et al. [34] had introduced the consistent hashing that guarantees that the distribution of the assigned identifiers on the P2P network is uniform with high probability; each node in the P2P network will be responsible for  $(1/N)$  keys where  $N$  is the number of nodes. In addition, during joining and leaving of peers, the minimum number of keys will be moved to new responsible nodes. Stoica et al. [3] proposed the virtual node mechanism to increase the fairness between the nodes. The virtual node looks like a single peer to the underlying DHT while each physical node can be responsible for more than one virtual node.

Rao et al. [35] used the advantage of splitting the load into virtual nodes to move a virtual node from any node to any other node in the system. Their solution provides a higher load-balancing in a heterogeneous P2P network through the use

of one of three schemes, One-to-One scheme, One-to-Many scheme, and Many-to-Many scheme. In the One-to-One scheme, each lightly-loaded node can periodically pick a random ID and then perform a lookup operation to find the node that is responsible for that ID. If that node is a heavily-loaded node, then a transfer may take place between the two nodes from the heavily-loaded node to the lightly-loaded node. In the One-to-Many scheme, the heavily-loaded node considers more than one lightly-loaded node at a time by maintaining directories that store load information about a set of lightly-loaded nodes in the system referred to as a pool. The last scheme is Many-to-Many, where the pool is only a local data structure used to compute the final allocation; no load is actually moved until the algorithm terminates. As such, the key difference between these three schemes is in the amount of information required to make the transfer decision.

Similarly, Bianchi et al. [36] presented an adaptive load balancing algorithm for DHT lookups which is based on prefix routing protocol, such as Pastry P2P protocol [20], with load-aware. The paper explains how to reorganize the routing table dynamically, and this done by choosing the nodes with the lowest load in order to offload the most heavily-loaded nodes. An extra enhancement to minimize the number of received requests at the nodes holding popular objects is caching the requested objects among the nodes along the path between the requesting nodes and the responsible nodes.



## CHAPTER 4

# INTENTIONAL DENIAL OF SERVICE TO DNS

### 4.1. Problem Description

As stated in the second chapter, the Resolvers need to go through the higher name servers (i.e., root and/or TLDs) to resolve the client's request. There are certain cases where the Resolvers fail to operate normally. In RFC1034 [4], a link failure, a gateway problem, or the unavailability of all the servers for a specific domain could cause a temporary failure of a Resolver. Alternatively, the name server could respond to the resolver's query with a *Refused* error code either because the Resolver is not welcome in this name server or the query is for restricted data [8]. Even the root nameservers could block a specific domain name due to a specific problem until the problem is solved [37].

The blocking behavior considered in the thesis is when the higher name servers refuse the Resolver's query intentionally. If the blocking happened to a specific domain name, the Internet will be inaccessible in the regions served by this domain name. Accordingly, the thesis addresses the problem of the higher level nameservers (i.e., root or TLDs nameservers) intentionally blocking the DNS traffic

coming from a specific Resolver that could represent a specific region. As shown in Figure 4.1, the root server responds to the queries send by Resolver A, Resolver C, and Resolver D. However, it receives Resolver B's queries but does not respond intentionally that make Resolver B in a blocking state.

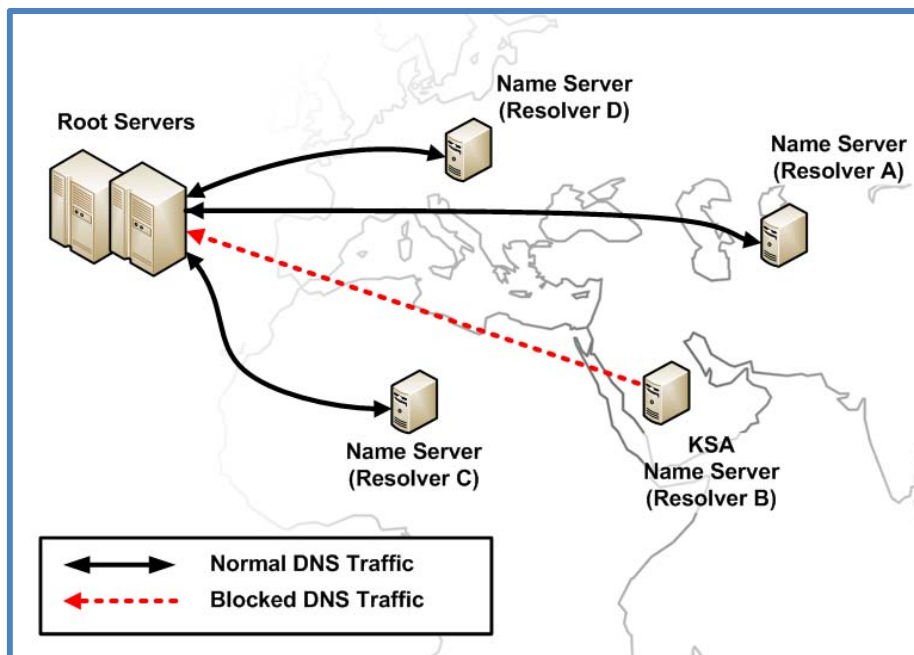


Figure 4.1: An illustration of the blocking behavior at KSA

## 4.2. Solution Objectives

To solve the problem of the intentional blocking of the DNS service from a specific region, the following objectives have to be considered in the solution:

- Providing an alternative path to resolve the DNS query if the existing path has failed or becomes blocked.
- The solution should be integrated with the existing DNS without any modification to the existing DNS standard.
- The added latency to resolve the DNS query through the alternative path should be minimal.
- Incoming DNS queries to the blocked server are distributed equally between the participating servers.
- The solution should be as robust as possible against multiple path blocking or failures.

## CHAPTER 5

### **DYNAMIC ROUND-ROBIN P2P**

#### **5.1. Introduction**

To solve the intentional DNS blocking, a simple forwarding of the DNS traffic to a neighbor DNS resolver will help. However, the sudden increase of the DNS traffic from the neighbor could trigger the higher DNS servers to investigate and block the neighbor as well. An alternative solution can use more neighbors so as to distribute the load among the participating neighbors. The communications between these neighbors could be established through a structured P2P network or through the use of a round-robin approach. Using the structured P2P network will create a scalable and stable network but with an extra latency. On the other hand, a round-robin approach provides a fast path to resolve queries but it will suffer when the network size is increased because of the need to maintain the routing information for every network node when a node joins or leaves the network. Another solution can be created by combining the scalability and stability of the structured P2P network and the load-balancing and the fast resolving of the

round-robin approach. Such a solution will be referred to as the Dynamic Round-Robin P2P solution.

The dynamic round-robin P2P solution is a modified version of Chord protocol where the stabilization process is done using Chord protocol and resolving lookup queries is done using round-robin scheme between the nodes in the routing table. The routing table in dynamic round-robin P2P is build from the unique nodes in the Finger table that updated in every stabilization process. Table 5-1 summaries the difference between the proposed solution and Chord protocol.

Table 5-1: The difference between Dynamic Round-robin P2P and Chord protocol

	Query Lookup	Table Update
<b>Chord</b>	Chord <ul style="list-style-type: none"> <li>• Successor List</li> <li>• Finger Table</li> </ul>	Chord <ul style="list-style-type: none"> <li>• Successor List</li> <li>• Finger Table</li> </ul>
<b>Dynamic Round-robin P2P</b>	Round-Robin <ul style="list-style-type: none"> <li>• Routing Table</li> </ul>	Chord <ul style="list-style-type: none"> <li>• Successor List</li> <li>• Finger Table</li> </ul>

## 5.2. Assumptions

- The blocking behavior is known by some mechanism.
- The higher nameservers are not blocking RRs of a specific region to the rest of the Internet.

- The DNS space is equally likely requested (uniformly distributed).
- The peer DNS servers participating in the proposed solution are trusted.
- From a performance point of view, the local DNS servers (i.e., Cache Resolvers) contributing in the system are high performance servers with reasonable bandwidth and homogenous resources (i.e., symmetric).
- Each peer can communicate with other peers through a direct or a virtual route.
- There is no disconnection between peers (i.e., no physical problems or configuration errors).
- The DNS RRs are not necessarily available in the responsible peer. The peer will resolve it from higher DNS servers through the existing DNS system.

### 5.3. Node States

There are five different states to any node in the simulator that used in this study; ACTIVE, BLOCKED, ISOLATED, DEPARTED, and FAILED. The simulator is described in details in CHAPTER 6. These states are divided into two sets; ALIVE and DEAD. The ALIVE nodes; i.e., ACTIVE, BLOCKED, and ISOLATED; are running during the simulation and have associated working functions. On the other hand, the DEAD; i.e., DEPARTED and FAILED; nodes are removed from the network and

no longer participate in the simulator. The explanation of the five different states follows.

1. ACTIVE:

- Default state for any node that joins the network.
- Considered as an ALIVE node.
- The node in this state can send queries, forward queries, and resolve queries.

2. BLOCKED:

- When an ACTIVE node discovers that it is blocked, it changes its state to BLOCKED.
- It sends a notification to its neighbors (i.e., Predecessor, and Successor) informing them that it is BLOCKED.
- Considered as an ALIVE node.
- The node in this state can send queries, as well as forward queries.
- The node in this state can NOT resolve queries.

3. ISOLATED:

- It is a special state when a large number of nodes failed to operate in the P2P network (> 70%). Thus some of the ACTIVE nodes will find out that all of their fingers and successors are DEAD, so it will change its state to ISOLATED.

- Considered as an ALIVE node.
- The node in this state can resolve queries.
- The node in this state can NOT send queries, nor can it forward queries.

#### 4. DEPARTED:

- Before an ALIVE node voluntarily departs from the network, it changes its state to DEPARTED.
- It sends a notification to its neighbors (i.e., Predecessor, and Successor) informing them that it is DEPARTED.
- Considered as a DEAD node.
- The node in this state can NOT send queries, forward queries, or resolve queries.

#### 5. FAILED:

- When an ALIVE node fails, it changes its state to FAILED.
- No notification is sent in this case.
- Considered as a DEAD node.
- The node in this state can NOT send queries, forward queries, or resolve queries.



A summary for these states are shown in Figure 5.1. The figure shows the state is either ALIVE or DEAD and what are its main functions (i.e., generating, resolving, and forwarding).

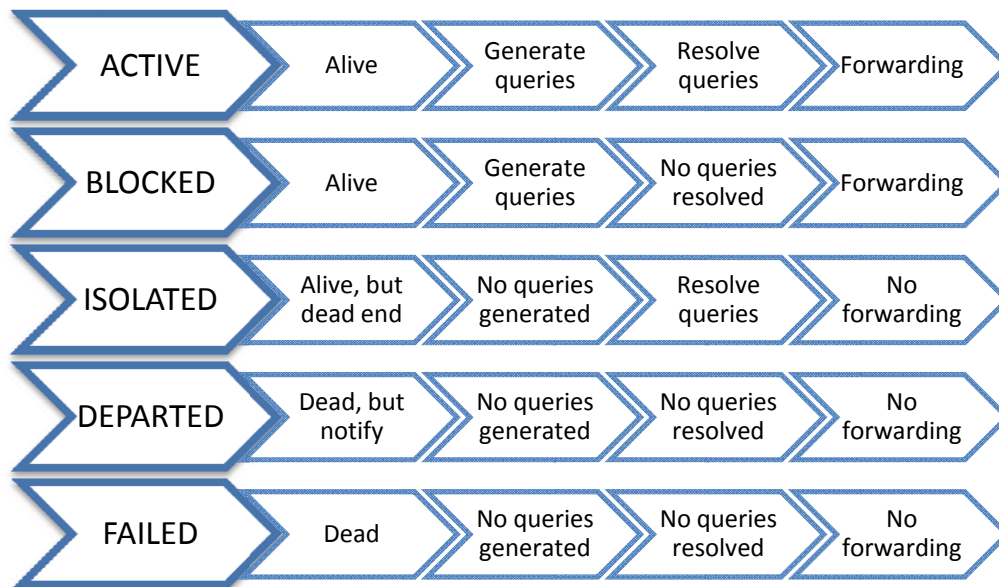


Figure 5.1: Node states

## 5.4. Proposed Solution

The proposed solution is based on the Chord protocol [3], and the round-robin approach. Thus, each resolver will have a list of peers; that built the routing table; to forward the query to in a round-robin fashion in addition to forwarding the query using the normal DNS lookup procedure. When the query is forwarded to the next hop, the next hop will resolve the query through the existing DNS lookup

procedure (i.e., checking the cache or asking the higher nameservers). However, when the next hop is BLOCKED, the query will not be resolved. To keep the routing table updated with ACTIVE peers and minimize the unresolved queries, there are two modifications to the normal round-robin approach; using Chord as the underlying layer and using an enhanced round-robin approach. Chord protocol is used as an underlay infrastructure to solve the scalability issue in the round-robin approach by updating (i.e., stabilizing) the list of peers comprising the finger table in the Chord protocol. The stabilization process is modified to support the scenario of a BLOCKED peer. An enhancement to the round-robin technique is to go through the entire finger table if there are FAILED nodes, which guarantees a successful lookup. A simple illustration of the proposed solution is shown in Figure 5.2 where the DNS server at KSA is participating in the P2P network with its trusted neighbors. If the DNS traffic that is requested from the KSA resolver is blocked from the root server, the KSA resolver can resolve its queries through the proposed P2P network. Assuming that the KSA resolver has Resolver A and Resolver D as the entries in its local routing table, then the KSA resolver will send the first query to Resolver A and the second query to Resolver D and continue in round-robin fashion. The solution will take care of the routing table if Resolver A state is FAILED or DEPARTED; i.e., the new routing table will include Resolver C and Resolver D.

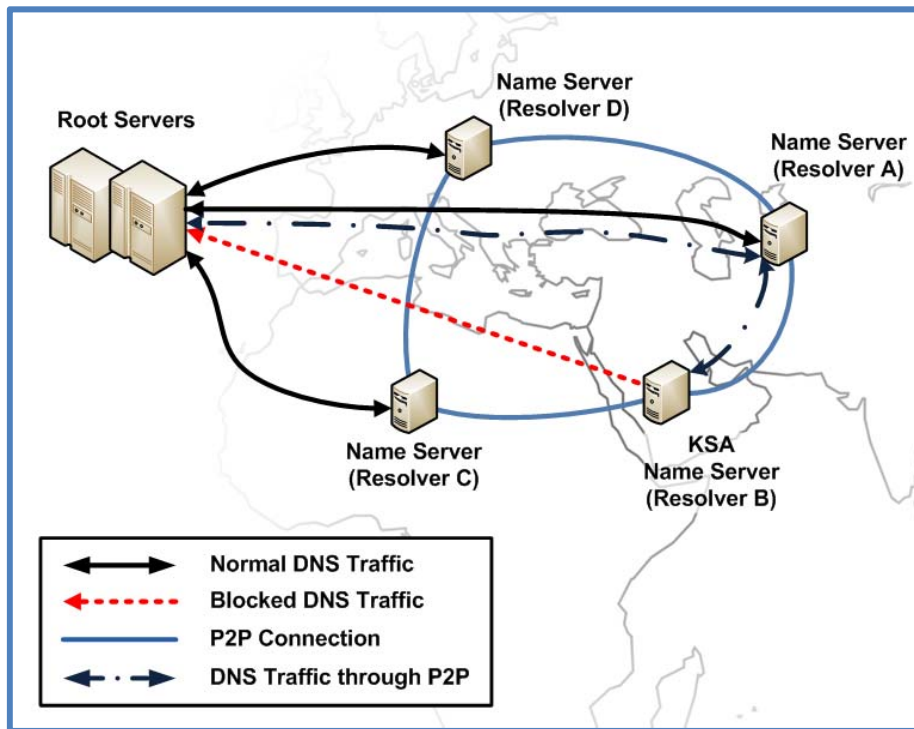


Figure 5.2: An illustration of the proposed solution at KSA

## 5.5. Methodology

To achieve the solution objectives stated in section 4.2, the following tasks are considered:

1. Review of literature of recent related work for the purpose of investigating different solutions for load balancing in P2P.

2. Implement Chord P2P simulator and verify its functionality with the results of Stoica et al.[3].
3. Modify the simulator to support an enhanced query forwarding technique and to account for the scenario of blocking some of the participating peers.
4. Compile the results and evaluate the proposed solution.

## **5.6. Modified Chord Protocol**

As stated in section 5.3, the proposed solution is partly based on a modified Chord protocol. There are three modifications to the Chord protocol that was introduced in [3]. These three modifications are explained in the following subsections.

### **5.6.1. Find Successor Procedure**

As shown in Figure 5.3, there is a new condition to the Find Successor Procedure stated in section 2.2.2 that checks if the entry in the Successor List is BLOCKED or DEAD. Since the BLOCKED node cannot resolve any query, it will not be used in the Find Successor procedure. However, it will be utilized in the Closest

Predecessor procedure to increase the possibility of reaching the closest node. The BLOCKED node does not cause a timeout since the BLOCKED node is still ALIVE.

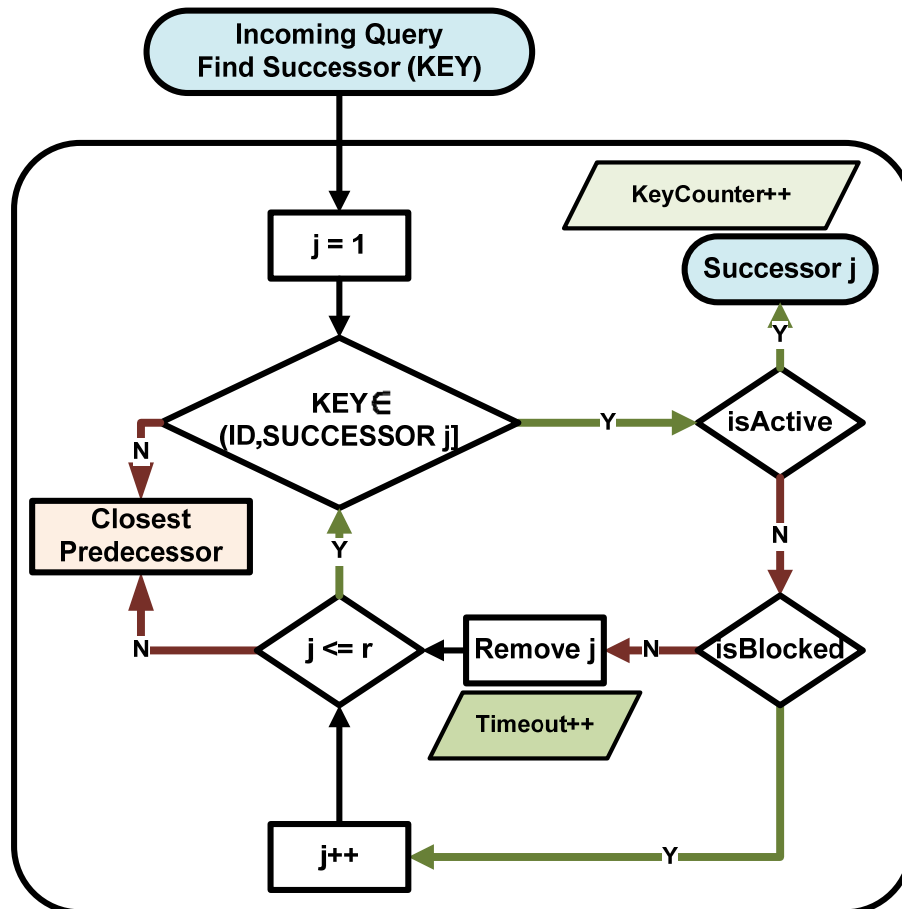


Figure 5.3: Modified Find Successor procedure

### 5.6.2. Closest Predecessor Procedure

There is nothing changed in the Closest Predecessor Procedure stated in section 2.2.2. However, the condition that checks if the entry is ALIVE or not will consider the BLOCKED node as an ALIVE node. As shown in Figure 5.4, each entry in the finger table or the successor list will be checked with the condition “ $Entry[k] \in (ID, KEY)$ ” if the node is either ACTIVE or BLOCKED; otherwise the node will be removed from the table.

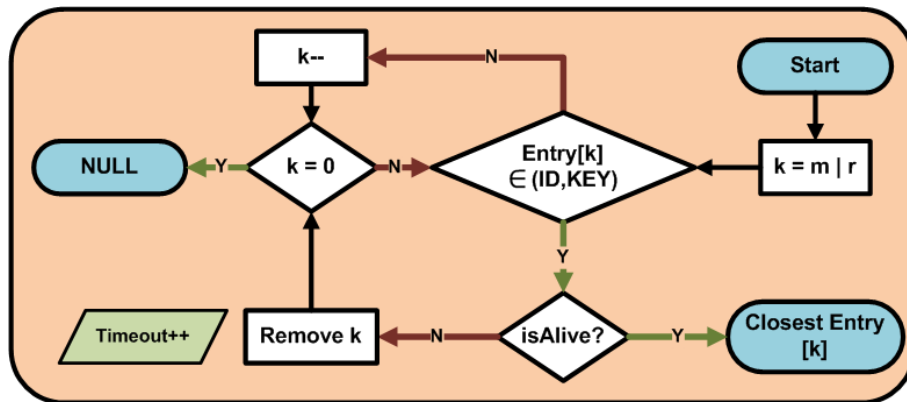


Figure 5.4: Modified Closest Predecessor procedure

### 5.6.3. Notify Neighbors

Notifying the predecessor and successor when a node voluntarily departs was suggested in [3] as stated in section 2.2.5. The same feature will be used when a node is blocked because the BLOCKED node is still ALIVE and it needs to inform its neighbors about its status that it could not resolve queries but it could help in finding the closest predecessor.

## **5.7. Modified Round-Robin**

The proposed solution utilizes a modification of the Chord protocol as described in section 5.6. Also, the proposed solution uses a modified round-robin approach as stated in section 5.3. The following subsections describe the modified round-robin approach, and, accordingly, the modifications to the DNS lookup procedure.

### 5.7.1. Lookup Window

To enhance the performance of round-robin, the algorithm of the round-robin will check a window of the routing table in each round. This window is configured by the node's administrator and it could be one entry resulting in the basic round-robin approach, portion of the routing table, or the complete table. This enhancement is applied only when the first entry in the window is DEAD or BLOCKED.

### 5.7.2. Lookup Procedure

Using the lookup window, the lookup procedure will start with the first entry in the window. If this entry is either DEAD or BLOCKED, the algorithm will remove that entry from the routing table. In addition, there will be a timeout when the entry is DEAD. If all entries in the window have been removed, the query message will be dropped. On the other hand, the message will be considered successfully resolved if there is an ACTIVE entry in the window. A pointer is used in order to help in determining the window's starting point in the next round. The pointer is incremented in a round-robin fashion after each check. Notice that the



window size will be adjusted whenever the routing table size is changed. This procedure is summarized in Figure 5.5.

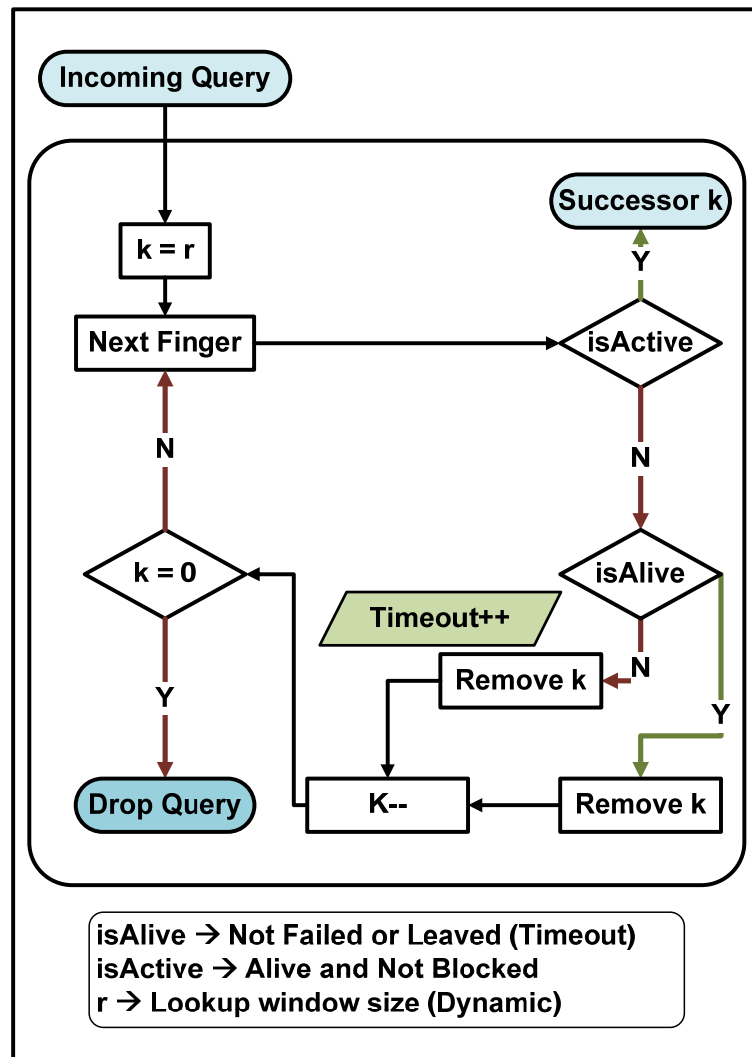


Figure 5.5: Lookup process in the modified round-robin

## 5.8. Complexity and Limitations

As stated in Stoica et al. [19], the time complexity of Chord protocol in stable network is  $O(\log_2 N)$ . This complexity will stay the same in the table update of the proposed solution because the Chord protocol is used as illustrated in Table 5-1. On the other hand, the time complexity of the query lookup is  $O(1)$  that is a result of using round-robin mechanism. The routing table in the proposed solution produced from the finger table that needs an  $m$  cycles to update the complete table where  $m$  is the finger table size. This behavior causes a delay for the stability of the network during the failure or blocking scenario. However, the time complexity will stay the same for low failed or blocked nodes ( $< 50\%$ ) which is similar to what stated in Stoica et al. [3]. With the current assumptions listed in section 5.2, the proposed solution has an advantage over the Chord protocol.

## CHAPTER 6

# SIMULATOR

There are different P2P simulators that vary in their characteristics. Examples of the P2P simulators include NS2, PeerSim, and P2PRealm[38]. NS-2 is a complex event-driven simulator built on C++ that can run a real network simulation with limited scalability [39]. PeerSim is a P2P Java-based simulator with good scalability but the P2P protocol to be simulated need to be written in modules to be run in this simulator. Even though there are some predefined protocols, the module and metrics need to be modified to fit with the proposed solution[40]. P2PRealm is another Java-based simulator that is efficient in studying neural networks but not suitable for the proposed solution [41]. Naicken et al. [42] stated that the percentage of custom simulators in the published P2P papers exceeds the percentage of well known P2P simulators.

The purpose of developing a new simulator is to study the behavior of the solution with the modified Chord protocol. The simulator is built using NetBeans IDE 6.8 with Java 1.6.0. The main modules of the simulator are core, simulation, analysis, and miscellaneous that are explained in the next sections, followed by a list of the simulator parameters. The analysis metrics are expressed in the third

section. Finally, a verification of the simulator with the results of Stoica et al. [3] is shown in the fourth section.

## 6.1. Simulator Modules

The simulator modules are divided into four sets of modules: core module, simulation module, analysis module, and miscellaneous module. The following sections explain these four sets of modules.

### 6.1.1. Core Modules

The core modules consist of 5 layers; i.e., modified Chord, modified round-robin, virtual node, real node, and network; and they are shown in Figure 6.1. The modified Chord protocol is the basic layer in the simulator. The main functions for this layer are joining of nodes to the network, stabilizing the network, and looking up queries. The modified round-robin layer overrides the query lookup that exists in the modified Chord layer with the round-robin's query lookup. The combination of these two layers is used as a P2P protocol to create a virtual node that is part of the real node as described in section 3.4. The network layer keeps track of the

nodes and their virtual nodes, and the keys that are explained in section 2.2.

Initializing the network and creating the query messages are done in the network layer.

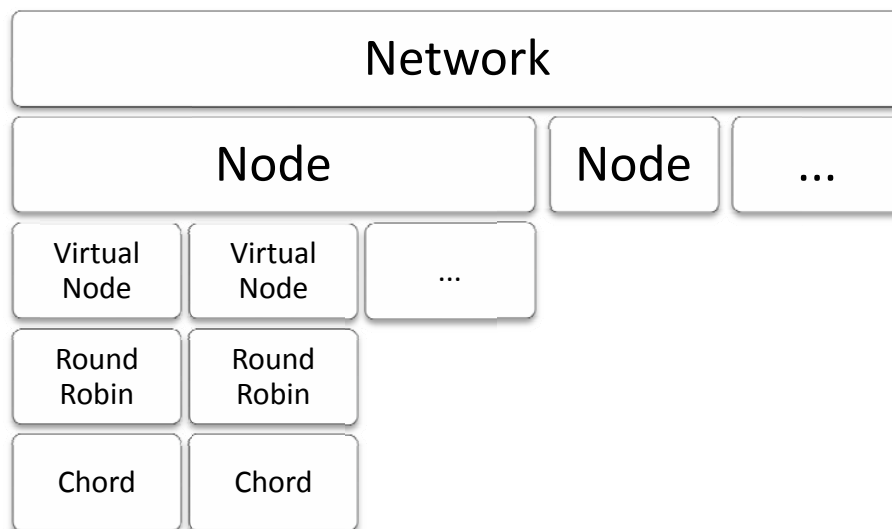


Figure 6.1: Core modules in the simulator

### 6.1.2. Simulation Modules

The simulator provides results for a single scenario or results for different scenarios, where a scenario differs by one or more parameters. For each scenario, a set of samples are executed to observe the effects of a specific parameter to the scenario; and to obtain accurate results, each sample is executed for a number of

iterations. In this section, the three main modules that are used to run the simulator are explained.

1. Simulator Module:

- At the beginning of each sample of the simulation run, the simulator module will set the new scenario parameters.
- After each sample (i.e., after finishing the entire iterations), this module will collect the statistics about this sample and provide a summary report.
- This module has the capability to run different scenarios in the same execution. So, it will repeat until finishing all the samples and scenarios.

2. Scenario Module:

- Using the scenario parameters that are set using the simulator module, this module will initialize the network according to these parameters.
- There are six types of events that need to be scheduled at the beginning of the scenario through the use of the scenario module. These events are Query Lookup, ALIVE Node Stabilization, New Node Joining, Alive Node Departing, ACTIVE Node Blocking, and ALIVE Node Failing.

- This module will be repeated for the number of iterations in the setting.

### 3. Scheduler Module:

- This module is responsible of creating, scheduling, and executing events.
- The simulator is designed to be an event-driven simulator. That means it will continue running until there are no more events or the simulation time has expired.
- Although the simulator supports executing the Node Blocking and Node Failing events during the simulation running time, the Node Blocking and Node Failing events are executed once at the beginning of the simulation only.

The workflow of the simulator is summarized in Figure 6.2. For each scenario, the scenario's parameters are set and the network settings are reset. The scheduler module will create the initial events and execute these events until the end of the simulation time. Without resetting the random variables, the same procedure will be repeated for X iterations. At the end of each sample, the analysis module will provide a summary result for this sample. After that, the simulator will

repeat the scenario with a different sample until covering all samples in this scenario.

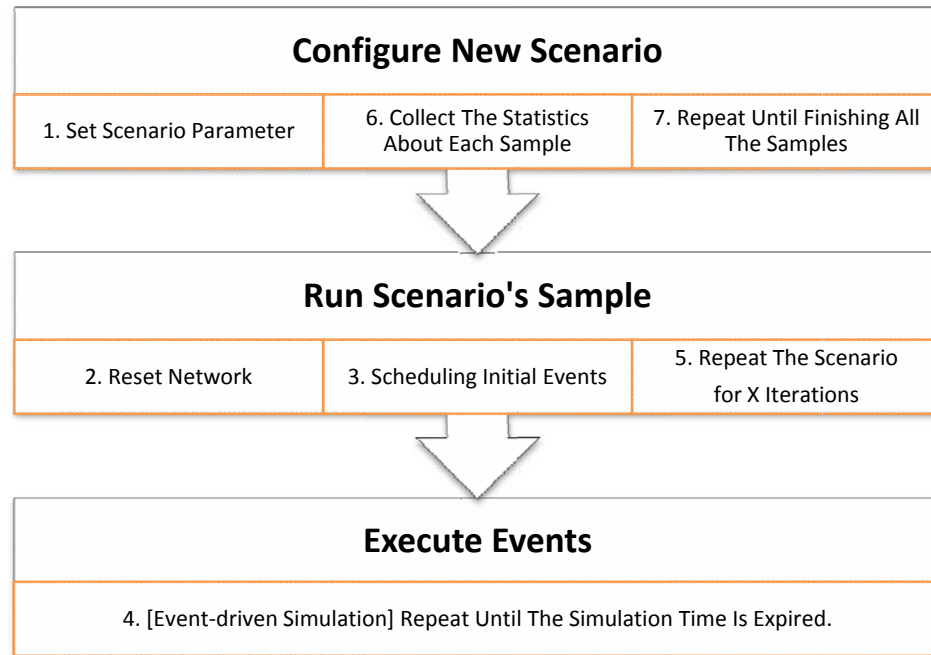


Figure 6.2: Simulation process

### 6.1.3. Analysis Modules

There are three modules; i.e., counter module, statistics module, analysis module; used to complete analyzing the behavior of the scenarios. The counter module is built in each node and message to gather the necessary information about the nodes and the messages. These counters are summarized in Table 6-1.



Table 6-1: Simualtor's counters descriptions

Counter	Location	Description
<b>message</b>	Real node	Count the total number of queries <b>generated</b> by the virtual node.
<b>key</b>	Real node	Count the total number of queries <b>resolved</b> by any virtual node.
<b>forward</b>	Real node	Count the total number of queries <b>forwarded</b> to any virtual node.
<b>forwardLookup</b>	Real node	Count the total number of queries <b>forwarded</b> to any virtual node and the query was a lookup query.
<b>stabilize</b>	Real node	Count the total number of <b>stabilization</b> done by the virtual node.
<b>stabilizeFailure</b>	Real node	Count the total number of <b>failed stabilization</b> done by the virtual node.
<b>fingerFailure</b>	Real node	Count the total number of <b>failed fix finger</b> done by the virtual node.
<b>timeout</b>	Message	Count how many times the message experience a timeout.
<b>numberOfHops</b>	Message	Count the number of hops until the message reaches the node that knows the successor for the message's key.

At the end of each iteration, the Analysis module would summarize these counters generated by the counter module into a statistical meaning using the Statistics module. These statistical expressions include average, first percentile, 99<sup>th</sup> percentile, and fairness. After the completion of a sample simulation, the Analysis module will present the average results of the analysis metrics defined in section 6.4.

#### 6.1.4. Miscellaneous Modules

Different modules are used to simplify the simulator and enhance the overall performance. These modules are summarized in Table 6-2

Table 6-2: Simulator's miscellaneous modules

Module	Description
<b>Definitions</b>	Contains the constant values and declarations.
<b>Configuration</b>	Contains the global variables.
<b>Tracer</b>	Used to display the result or the debugging messages during the simulation time in the screen or save them to a file.
<b>Timers</b>	To track the performance of the simulator.
<b>Utilities</b>	Useful methods that are used throughout the simulator; e.g., searching.

## 6.2. Simulator's Assumptions

- During the lookup query process, there are no nodes that are BLOCKED, DEPARTED, or FAILED before resolving the query.
- Propagation delay is not considered.
- There is only one lookup query resolved at any moment in time.
- At any time, the messages that are exchanged between the nodes belong to the same transaction.

- Node failure is not allowed during the study of a blocking event or a joining/leaving event.
- The simulator is only concerned with studying the performance of finding the responsible node. The RRs are not cached. Thus, no actual RRs movement takes place when RRs responsibility is moved between nodes.

### 6.3. Simulator Parameters

The simulator uses a number of parameters and variables including general parameters, random variables, network parameters, and node parameters. The following sections describe such parameters and variables.

#### 6.3.1. General Parameters

- **idSize**: identifies the number of bits in the node's ID (or key's ID). It has to be large enough to prevent the possibility of duplicate IDs.
- **simulationEndTime**: sets when the simulator should end in case the scheduler did not stop.

- **timeout**: when a message is sent and there was no response from the destination, the sender will wait for the timeout value before declaring that the destination virtual node is DEAD.
- **fileOutput**: enables writing up the tracing or the result to a file.
- **screenOutput**: enables writing up the tracing or the result to the screen.
- **LastKeyIndex**: defines the maximum number of queries running during a single iteration.
- **isContinues**: if it is enabled then the simulator will continue running until the simulationEndTime.
- **StabilizerEnabled**: used to enable the stabilization process. It is useful when testing the Chord protocol without stabilization.

The default values for these parameters are shown in Table 6-3.

Table 6-3: Default values for simulator's general parameters

Parameter	Default Value	Unit
<b>idSize</b>	60	bits
<b>simulationEndTime</b>	3,600,000	milliseconds
<b>timeout</b>	500	milliseconds
<b>fileOutput</b>	True	-
<b>screenOutput</b>	True	-
<b>LastKeyIndex</b>	10,000	queries
<b>isContinues</b>	False	-
<b>StabilizerEnabled</b>	True	-

### 6.3.2. Random Variables

There are different random variables used in the simulator as listed in Table 6-4. These random variables are reset once to their default values in the first iteration of each sample. The default values could be configured to be the same value for all random variables. The default values of the seed in Table 6-4 are picked randomly.

Table 6-4: Simualtor's random variable descriptions and default seeds

Random Variable	Distribution	Description	Seed
<b>nodeRV</b>	Uniform	Used to generate the ID for a new virtual node.	123
<b>keyRV</b>	Uniform	Used to generate a new key.	43
<b>queryKeyRV</b>	Uniform	Used to pick the key index for a new query message.	23
<b>querySourceRV</b>	Uniform	Used to pick the virtual node index for a new query message.	4321
<b>queryTimeRV</b>	Exponential	Used to generate the inter-arrival time for the query message.	543
<b>failureRV</b>	Uniform	Used to set the failure probability of the node.	53
<b>blockingRV</b>	Uniform	Used to set the blocking probability of the node.	61
<b>stabilizingRV</b>	Uniform	Used to generate the inter-arrival time for the stabilization process.	91
<b>joinRV</b>	Uniform	Used to generate the inter-arrival time for a new node to join.	531
<b>nRV</b>	Uniform	Used to pick the virtual node index for a new node joining.	771

Random Variable	Distribution	Description	Seed
<b>departRV</b>	Uniform	Used to generate the inter-arrival time for an ALIVE node to leave.	351
<b>IRV</b>	Uniform	Used to pick the virtual node index that will leave.	791

### 6.3.3. Network Parameters

- **Initial Network Size (N):** the total number of real nodes that the scenario starts with.
- **lookupProtocol:** can be set to either Modified Chord or Dynamic Round-Robin P2P
- **numberOfKeys:** the key space.
- **sourceTraffic:** identifies whether the lookup queries generated are from a single source (i.e., a specific virtual node) or from a random source (i.e., randomly selected virtual node).
- **Iterations:** total number of iterations per sample.

### 6.3.4. Node Parameters

- **Virtual Nodes:** identifies the number of virtual nodes per real node.

- **Blocking Percentage:** refers to the probability of the real node to be blocked.
- **Failure Percentage:** refers to the probability of the real node to fail.
- **Churn Rate:** identifies the number of nodes joining and leaving per second.
- **Stabilization Period:** specifies the duration of the period through which each node runs the stabilization process. This period is uniform between [15, 45] seconds.
- **Successor List:** refers to the size of the successor list.
- **Lookup Window:** specifies the size of the lookup window that is used in the modified round-robin approach.

## 6.4. Performance Metrics

The following subsections describe the analysis metrics used for the comparison between the Chord protocol and the proposed round-robin P2P approach. Note that the unit of the analysis metric, if any, appears between parentheses after the name of the analysis metric.

1. **Load (keys/node):** It is the average number of keys per node that only appear in the lookup messages.

$$load_{successful\ lookup\ messages} = \frac{\sum resolved\ keys}{\sum real\ node\ resolvers}$$

2. **Traffic (message/node):** It is the average number of messages per node.

These messages are any forwarded queries.

$$traffic = \frac{\sum forwarded\ messages}{\sum real\ nodes\ participated}$$

3. **Fairness:** It is used to reflect how uniformly the query resolving is distributed among the nodes that will be referred to as “load fairness”. Also, fairness is used to identify the uniformity of the number of messages forwarded by each node. The later usage of fairness will be referred to as “traffic fairness”.

$$fairness = \frac{(\sum messages\ per\ real\ node)^2}{\sum real\ nodes \times \sum (messages\ per\ real\ node)^2}$$

In the equation, the messages in the load fairness are the resolved queries that only appear in the lookup messages and in the traffic fairness are any forwarded queries.



4. **Percentage of Failed Lookup:** Percentage of dropped queries out of total queries sent.

$$\text{Percentage of failed lookup} = 100 \times \frac{\sum \text{dropped lookup}}{\text{Total number of lookup queries}}$$

5. **Path Length (hops):** It is the number of hops needed to identify the resolver (i.e., successful lookup).

$$\text{Path length} = \frac{\sum \text{path length}}{\sum \text{successful lookup messages}}$$

6. **Timeout (seconds):** It is the average timeout when a node does not respond (i.e., either failed or departed) in a successful lookup.

$$\text{timeout} = \frac{\sum \text{timeout}}{\sum \text{successful lookup messages}}$$

## 6.5. Verification

The Chord behavior in the simulator was verified by reproducing the results of Stoica et al. [3]. The same assumptions in the paper are used in the simulator to reproduce four scenarios that will be discussed subsequently. In general, the simulator has the same behavior as the Chord protocol presented in Stoica et al. paper [3].

### 6.5.1. Scenario 1: Load Balance Scenario

In a 10,000 nodes Chord network, traffic is generated by using  $k \times 100,000$  unique keys, where  $1 \leq k \leq 10$ . As shown in Figure 6.3, the average load in each node equals

$$\frac{\sum \text{keys}}{\text{the network size}} = \frac{100,000 \times k}{10,000} = 10 \times k$$

Also, the first and 99th percentiles are shown in Figure 6.3. As expected the 99th percentile increases proportionally with  $k$ . By increasing the number of virtual nodes per real node in the simulator, the 99th percentile decreases and the first percentile increases as reported in [3]. Both Stoica et al.'s results and the simulator results are shown in Figure 6.4.

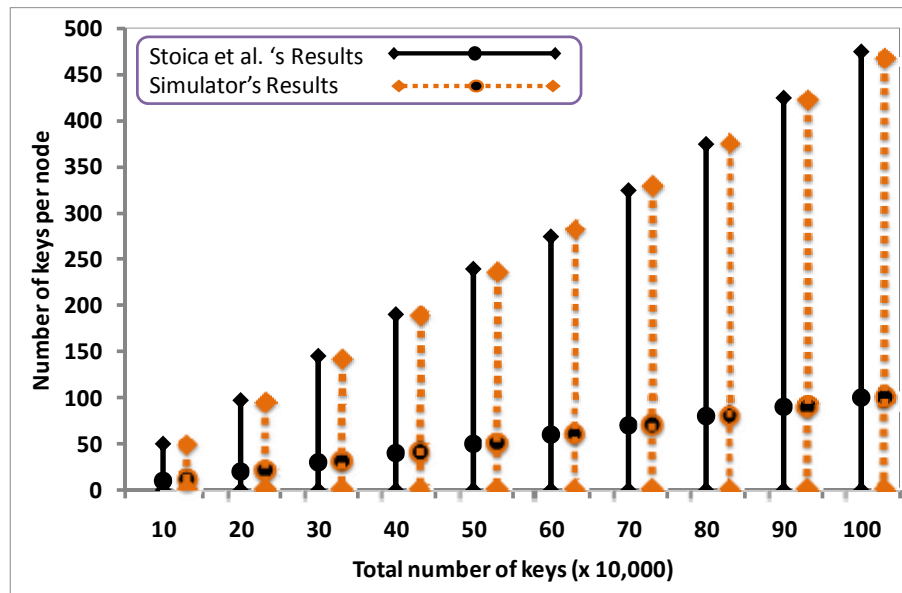


Figure 6.3: Load Balance Scenario

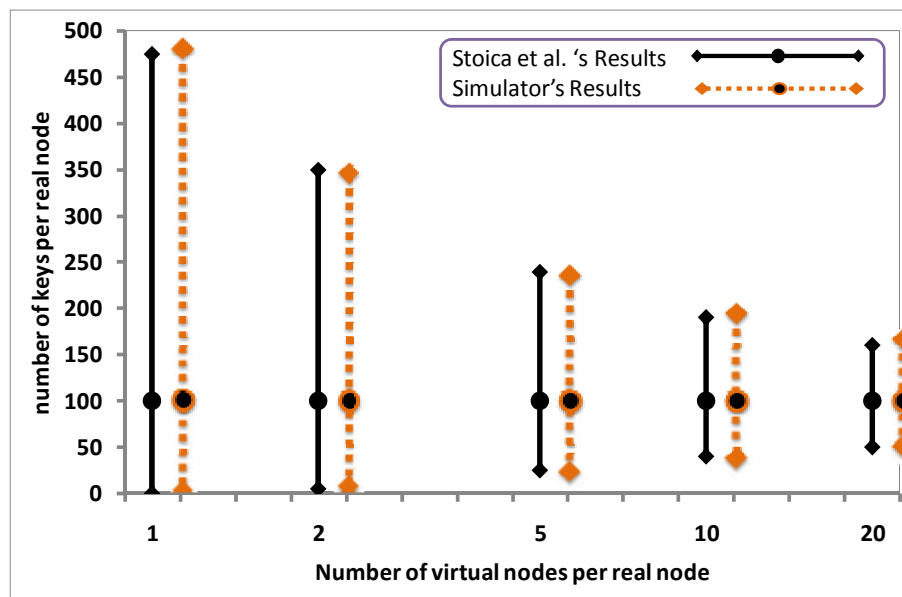


Figure 6.4: Load Balance Scenario with different virtual nodes

### 6.5.2. Scenario 2: Path Length Scenario

The results presented in Figure 6.5 show the number of hops needed to know the successor for a key by calculating the hops for the traffic generated in  $2k$  nodes and  $100 \cdot 2k$  keys, where  $3 \leq k \leq 14$ . The difference in the 1st percentile when the network size is less than 100 may be attributed to having some keys belonging to the same node that generates the traffic that results into no hops is required. This specific case was not clarified by Stoica et al. [3].

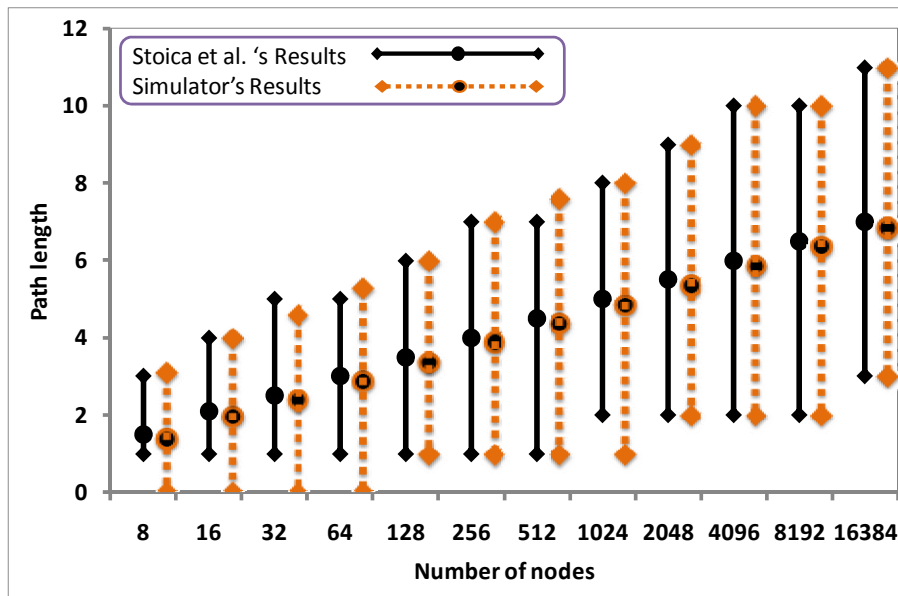


Figure 6.5: Path Length Scenario

### 6.5.3. Scenario 3: Nodes Failure Scenario

In this scenario, 10,000 random keys are queried after failing  $10 \cdot k\%$  nodes in the system, where  $0 \leq k \leq 5$ . The overall trend and the average of the lookup timeout results and the lookup path length results are the same for both Stoica et al. and the simulator as shown in Figure 6.6 and Figure 6.7. However, there are some differences between Stoica et al.'s results and the simulator results with respect to the first and the 99th percentile. This could be caused by missing parameters that was not stated in Stoica et al. paper [3].

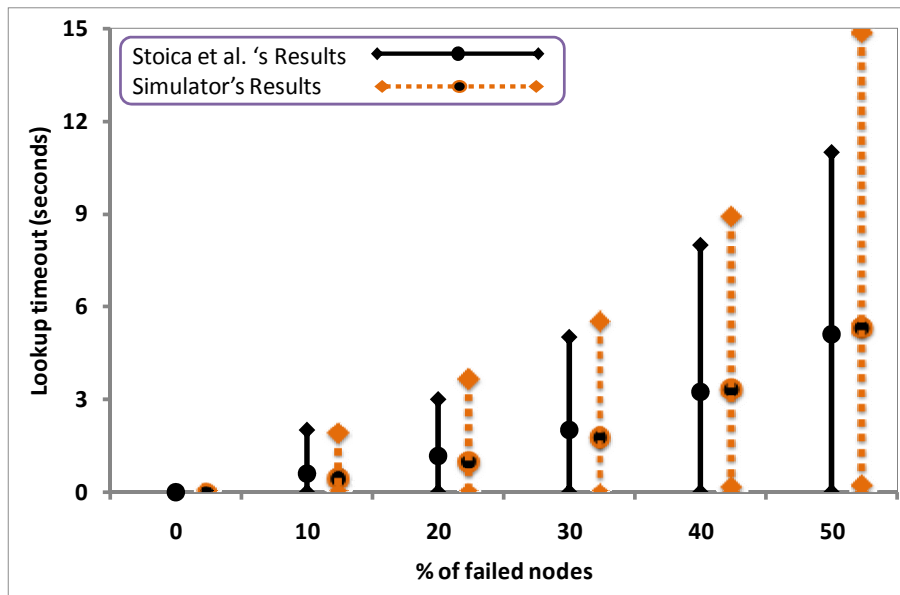


Figure 6.6: Nodes Failure Scenario – Timeout

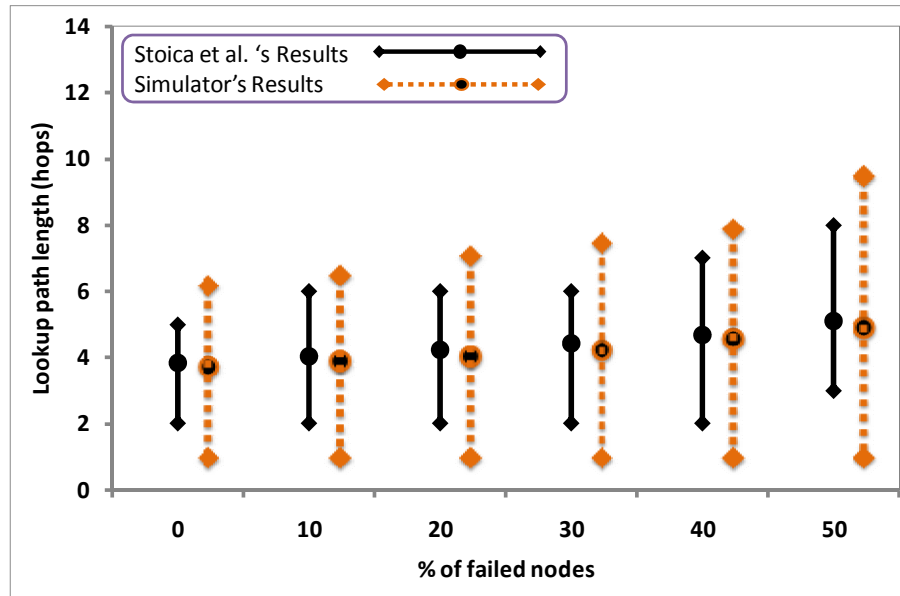


Figure 6.7: Nodes Failure Scenario - Path length

#### 6.5.4. Scenario 4: Joining and Leaving Scenario

Similar to scenario 3, the overall trend and the average of the lookup timeout results and the lookup path length results are the same for both Stoica et al. and the simulator as shown in Figure 6.8, Figure 6.9, and Figure 6.10 with some differences in the 99th percentile. These differences could be caused by missing parameters or procedures that were not stated by Stoica et al.. On the other hand, the number of failed lookup in Figure 6.10 is less than what appears by Stoica et al.. However, because both results are relative to 10,000 lookup messages, the numbers obtained in both Stoica et al. and the simulator are considered negligible.

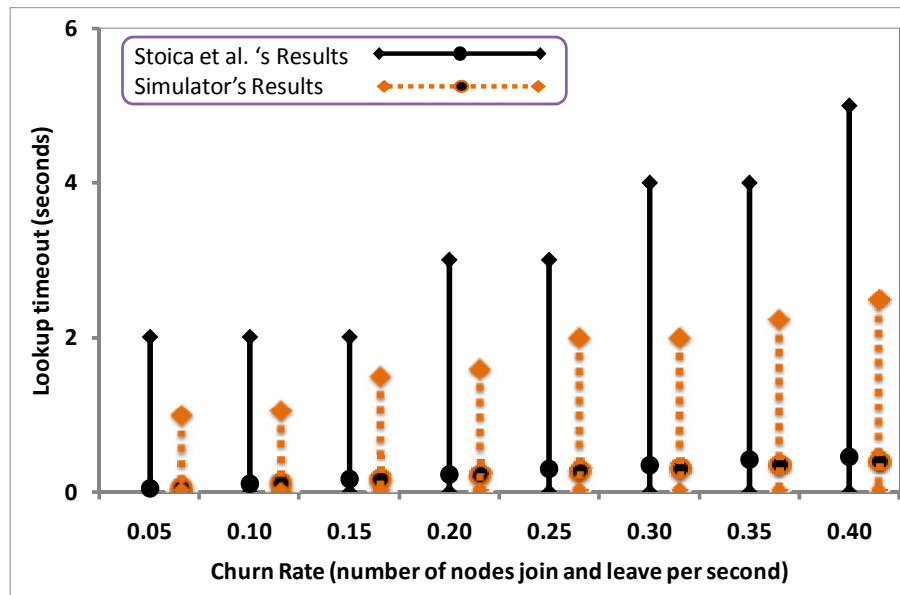


Figure 6.8: Joining and Leaving Scenario - Timeout

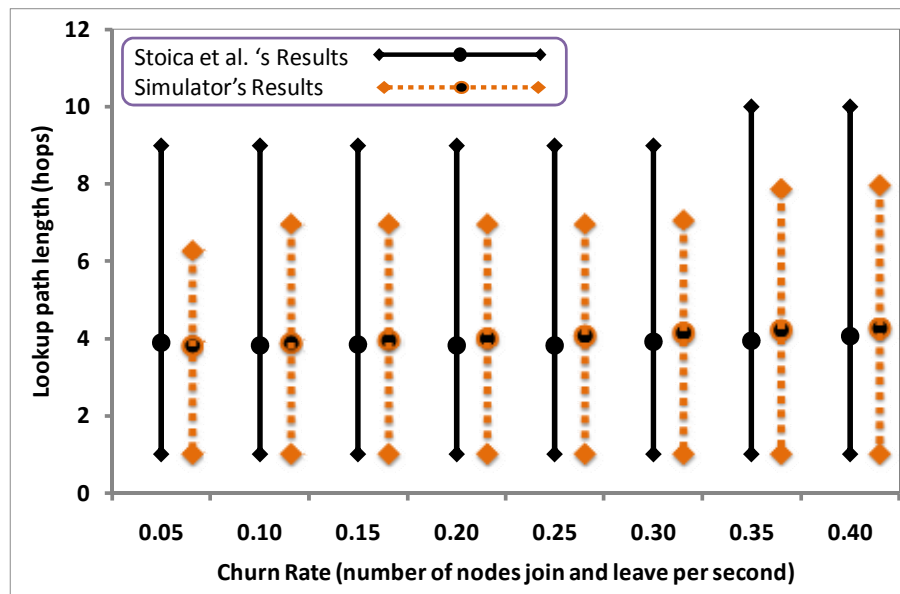


Figure 6.9: Joining and Leaving Scenario - Path length

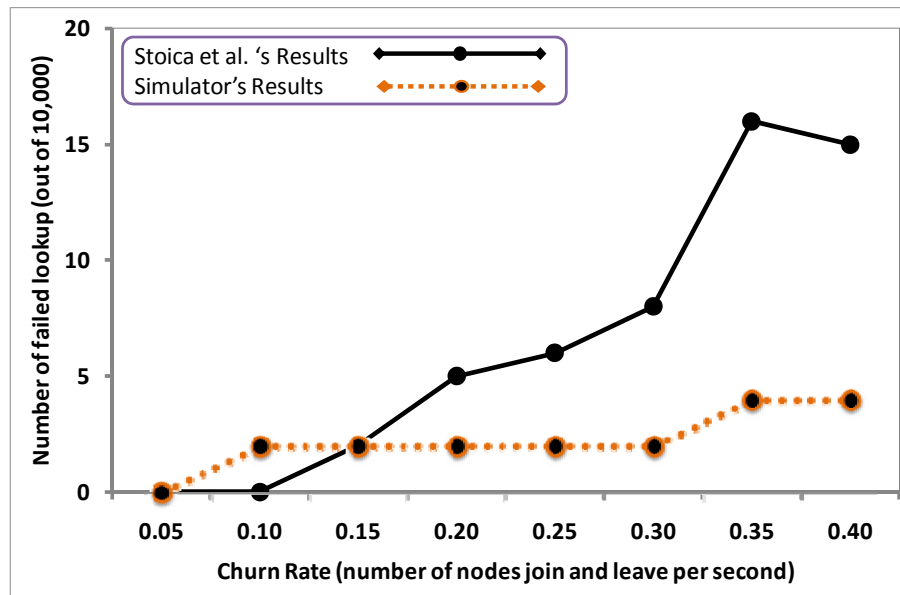


Figure 6.10: Joining and Leaving Scenario - Lookup failure



## CHAPTER 7

### RESULTS AND ANALYSIS

In this chapter, three scenarios are simulated and analyzed; Blocking, Failure, and Joining and Leaving. Each scenario consists of four network configurations with each configuration repeated with different number of virtual nodes per real node. The different numbers of virtual nodes considered are one virtual node (v1), four virtual nodes (v4), and 16 virtual nodes (v16). The four configurations examined are:

1. Chord-20: it is the basis of the other configurations and is used to update the routing information of the other configurations. The Successor List size in Chord-20 is  $2 \times \log_2 N$ . So, with 1,000 nodes network and one virtual node per real node, the size equals 20. Chord-20 was picked for implementation since it was preferred in Stoica et al. paper.[3]
2. Round-1: changing the lookup process from Chord to round-robin with the lookup window size set to one.
3. Round-5: changing the lookup process from Chord to round-robin with the lookup window size set to half the routing table.

4. Round-10: changing the lookup process from Chord to round-robin with the lookup window size set to the complete routing table.

## 7.1. Scenario Parameters

Table 7-1 summarizes the common parameters that are used in all scenarios. If there is any change in any parameter, it will be clarified in the scenario description.

Table 7-1: Scenarios' default parameters

Parameter	Value
Iterations	10 iterations per sample
Initial Network Size	1,000 real nodes
Virtual Nodes	1, 4, 16 virtual node(s) per real node
Successor List Size	$2 \times \log_2 N$ ( $N$ = total virtual nodes in the network)
Lookup Window Size	1, $\frac{1}{2}$ * routing table size, full routing table size
Query Message	A key and a live source are chosen randomly
Number of Queries	10,000 lookup queries
Query Inter-arrival Time	Exponential with mean of 50 milliseconds
Timeout	500 milliseconds
Stabilization Period	Uniform [15, 45] seconds

## **7.2. Blocking Scenario**

### **7.2.1. Objective**

The objective of the blocking scenario is to study the behavior of resolving the queries with the proposed solution when there are blocked nodes in the network. The behavior is studied for both the modified Chord and the round-robin approach.

### **7.2.2. Scenario Description**

After a network of 1,000 real nodes is stable, the state of a portion of the network is changed from ACTIVE to BLOCKED. This portion is varied from 0% to 90%, increasing 10% in each sample. Then, uniform random queries are generated using the default parameters.

### 7.2.3. Scenario Results

#### 7.2.3.1. Lookup Failure

There is no lookup failure in this scenario when using Chord-20 as shown in Figure 7.1. That is because the blocked node will notify its neighbor. Accordingly, the neighbor updates its successor list. On the other hand, the other configurations depend on the routing table that is only updated during the stabilization process (i.e., finger table updating). Also, a sufficient number of queries is needed to detect and remove a BLOCKED node from the routing table. Hence, Round-1 has the highest lookup failure percentage.

By increasing the total number of queries in the network from 10,000 queries to, for example, 100,000 queries, each node will generate enough queries to remove the BLOCKED nodes from the routing table. Accordingly, the percentage of lookup failure reduces significantly for most of the configurations as shown in Figure 7.2. However, Round-1 still suffers from the worst lookup failure.

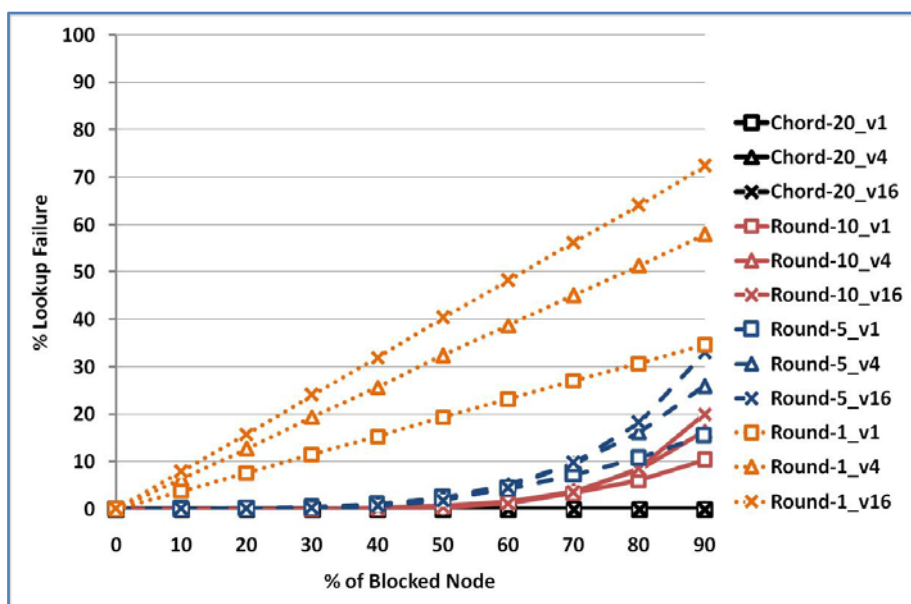


Figure 7.1: Blocking Scenario - Lookup failure percentage with 10,000 queries

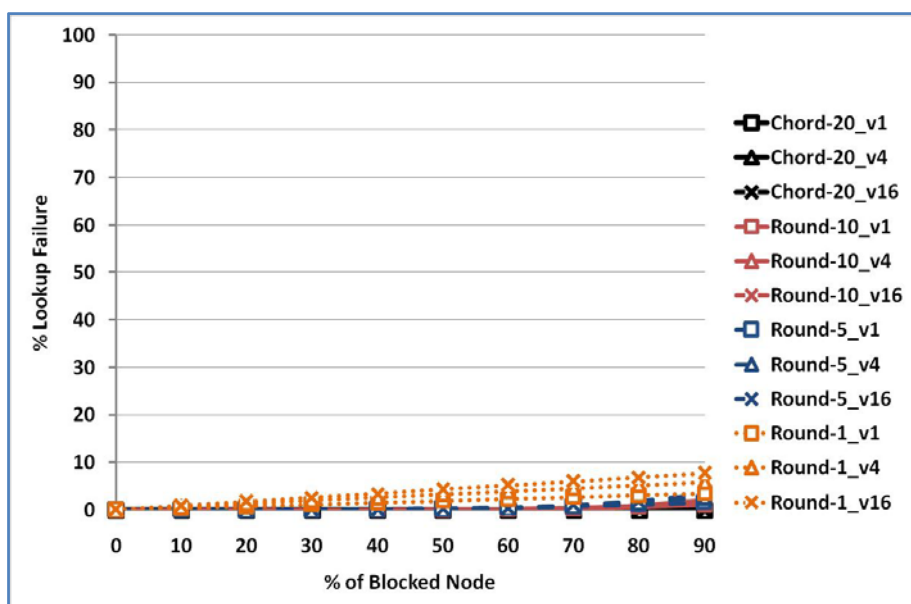


Figure 7.2: Blocking Scenario - Lookup failure percentage with 100,000 queries

### 7.2.3.2. Load Balance and Fairness

The load balance results are shown in Figure 7.3 and Figure 7.4. Because Chord-20's successor lists are updated as a result of the notification sent by the BLOCKED node, the average number of keys per node matches with the ideal case that is described in the following equation:

$$\text{average number of keys per node} = \frac{\text{total number of keys (queries)}}{\text{total number of ACTIVE nodes}}$$

The other configurations have extra load because of the lookup failure.

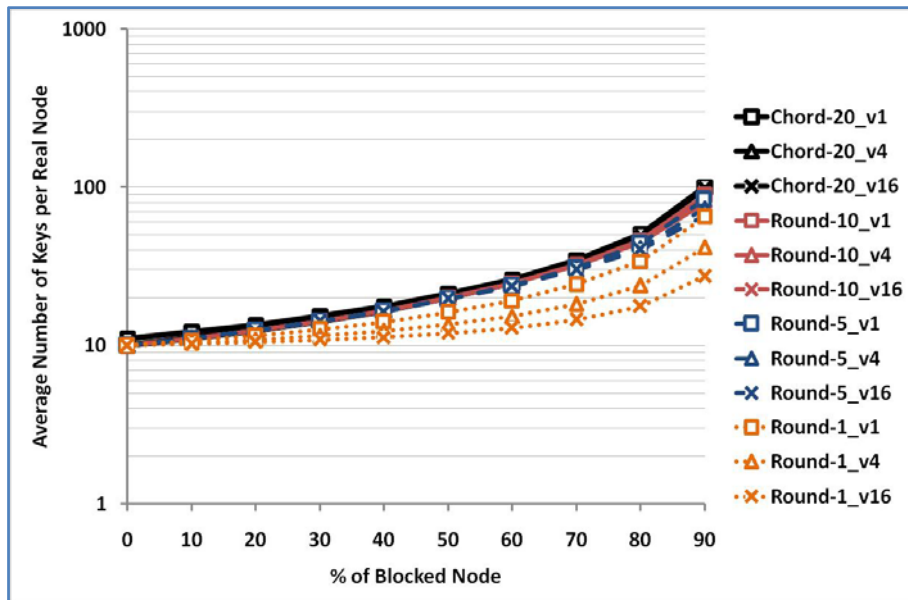


Figure 7.3: Blocking Scenario - Average load between the resolvers

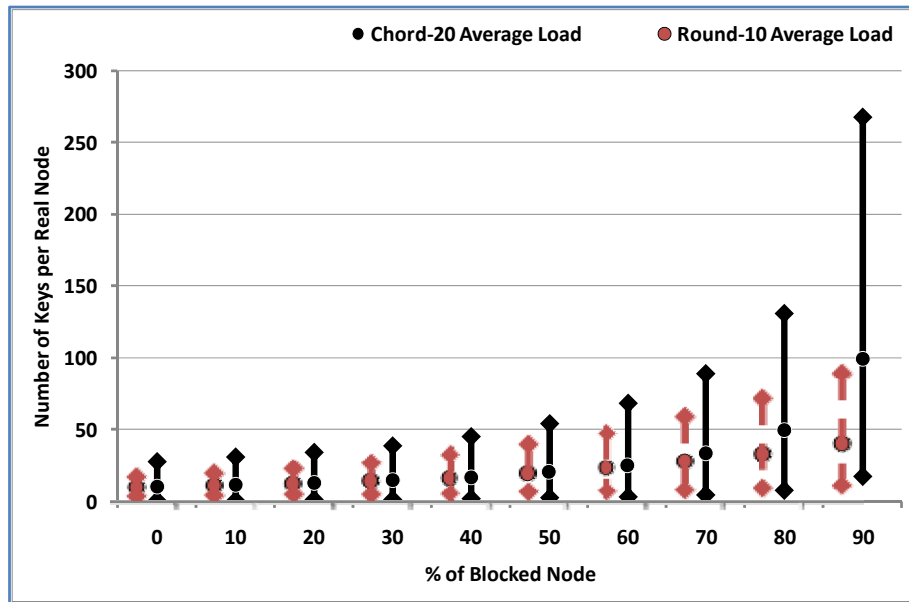


Figure 7.4: Blocking Scenario – The average, 1<sup>st</sup> and 99<sup>th</sup> percentiles, load between the resolvers in Round-10 and Chord-20 with four virtual nodes

On the other hand, it is observed in Figure 7.5 and Figure 7.6 that round-robin provides higher fairness between the nodes than Chord-20 due to its natural behavior. The low fairness of Round-1 is due to the way the finger table is built, which does not guarantee that the fingers will be distributed equally.

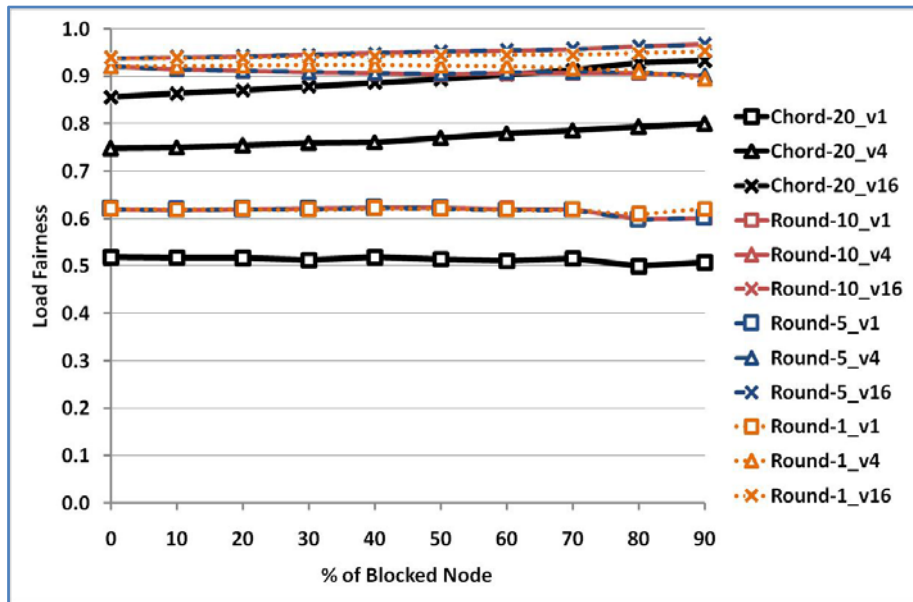


Figure 7.5: Blocking Scenario - Load fairness between the resolvers with 10,000 queries

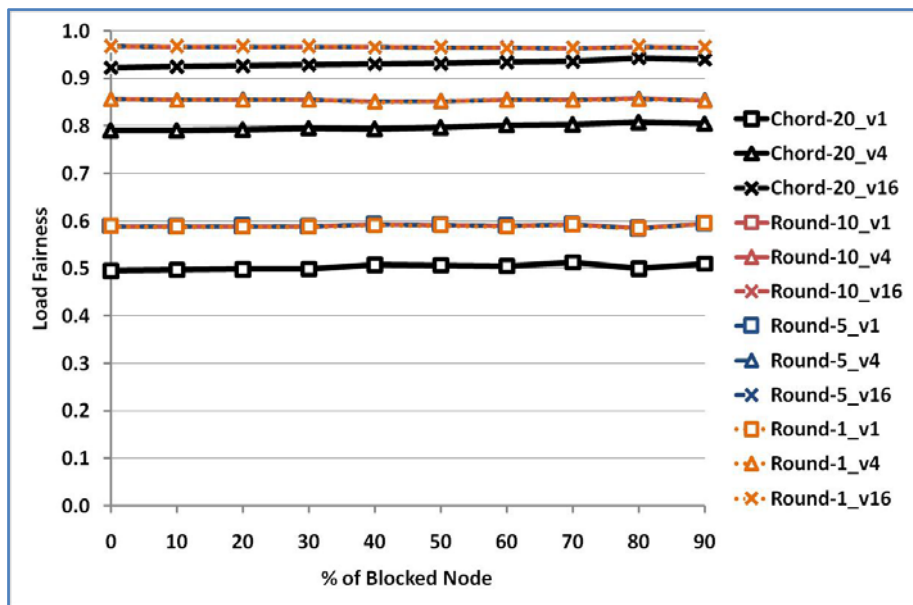


Figure 7.6: Blocking Scenario - Load fairness between the resolvers with 100,000 queries



### 7.2.3.3. Successful Lookup Path Length

Round-robin does not forward the queries to intermediate nodes, so there is no extra hop to resolve the queries as noticed from Figure 7.7. In Chord-20, the path length follows the following equation provided in [3]:

$$\text{Path Length} = \frac{\log_2 N}{2} - \frac{\log_2 r}{2} + 1$$

where  $N$  = network size, and  $r$  = virtual nodes per real node

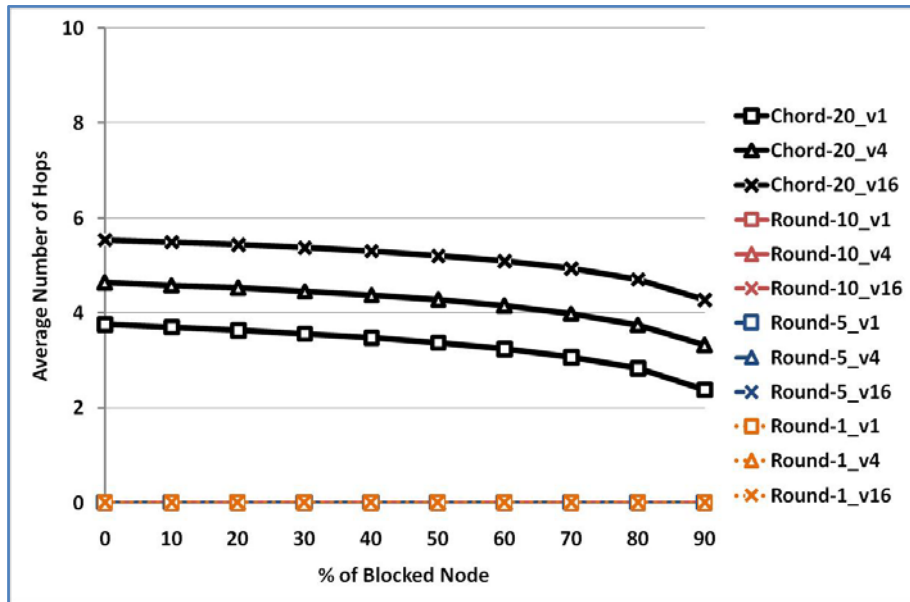


Figure 7.7: Blocking Scenario - Successful lookup path length

#### 7.2.3.4. Successful Lookup Timeout

The BLOCKED nodes are still considered as live nodes, which results in no timeout as reflected in Figure 7.8.

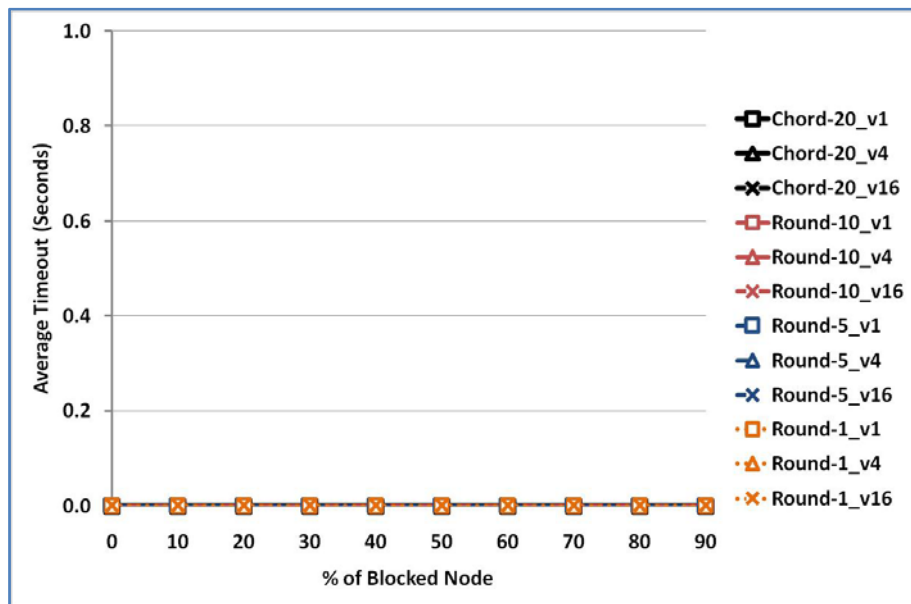


Figure 7.8: Blocking Scenario - Successful lookup timeout

#### 7.2.3.5. Traffic Balance

For each configuration, the average number of messages increases exponentially as the number of virtual nodes increases as is apparent in Figure 7.9.

This mainly happened due to the additional traffic introduced by the stabilization

process which increases when the number of virtual nodes increases. The difference between Chord-20 and the other configurations is due to the fact that the other configurations have no forward messages during the lookup process.

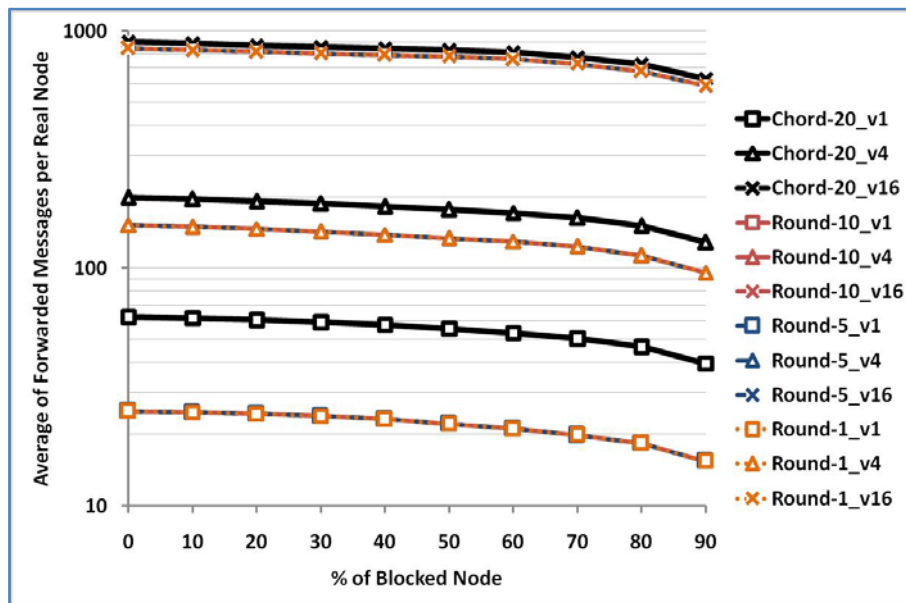


Figure 7.9: Blocking Scenario - Average traffic between the nodes

Furthermore, as shown in Figure 7.10, the fairness between the nodes drops due to the updating of the finger table during the stabilization process. In the situation when the stabilization process completely updates the finger table, there will be no BLOCKED nodes in the table. This causes the BLOCKED nodes to generate queries only and to stop forwarding the messages. This specific scenario is shown in Figure 7.11.

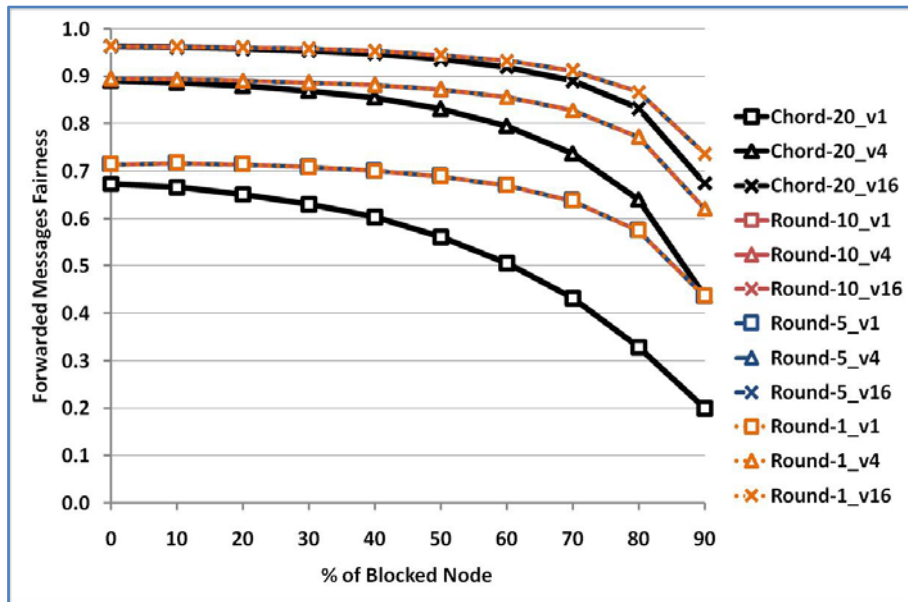


Figure 7.10: Blocking Scenario - Traffic fairness between the nodes with 10,000 queries

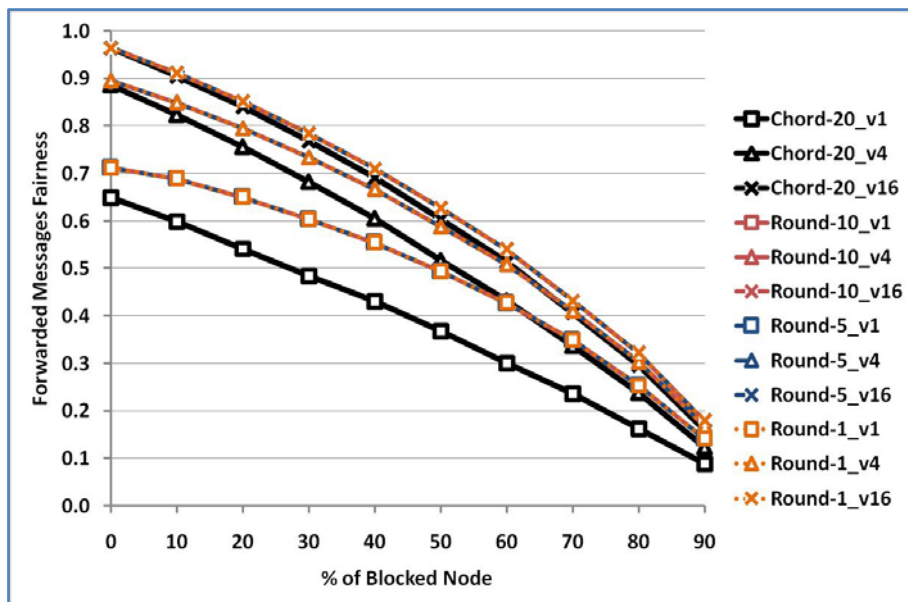


Figure 7.11: Blocking Scenario - Traffic fairness between the nodes with 100,000 queries

#### 7.2.4. Observations

The following observations can be made with respect to the previous results:

- There is no significant difference between the round-robin configurations except for the lookup failure.
- Increasing the virtual nodes has a bad influence on the round-robin configurations when considering lookup failure and traffic.
- When the blocking is less than 50% (i.e., normal operating conditions), Round-10 with four virtual nodes has better performance than Chord-20 with respect to lookup failure, traffic, and fairness.
- The difference between four and sixteen virtual nodes in Chord-20 could be considered negligible.

## **7.3. Failure Scenario**

### **7.3.1. Objective**

The objective of the failure scenario is to study the behavior of resolving the queries with the proposed solution when there are failed nodes in the network. The behavior is studied for both the modified Chord and the round-robin approach.

### **7.3.2. Scenario Description**

After a network of 1,000 real nodes is stable, the state of a portion of the network is changed from ACTIVE to FAILED. This portion is varied from 0% to 90%, increasing 10% in each sample. Then, uniform random queries are generated using the default parameters.

### **7.3.3. Scenario Results**

#### **7.3.3.1. Lookup Failure**

Chord protocol is a stable protocol whenever the successor list is accurate. This point is the main reason for using the successor list during node failure. The

probability that all the nodes in the successor list are failed decreases by increasing the successor list size. So, finding the successor of the key will be possible since there will be at least one active successor in each node. However, as the percentage of failed nodes increases, the network stability is decreased and the dropped messages are increased.

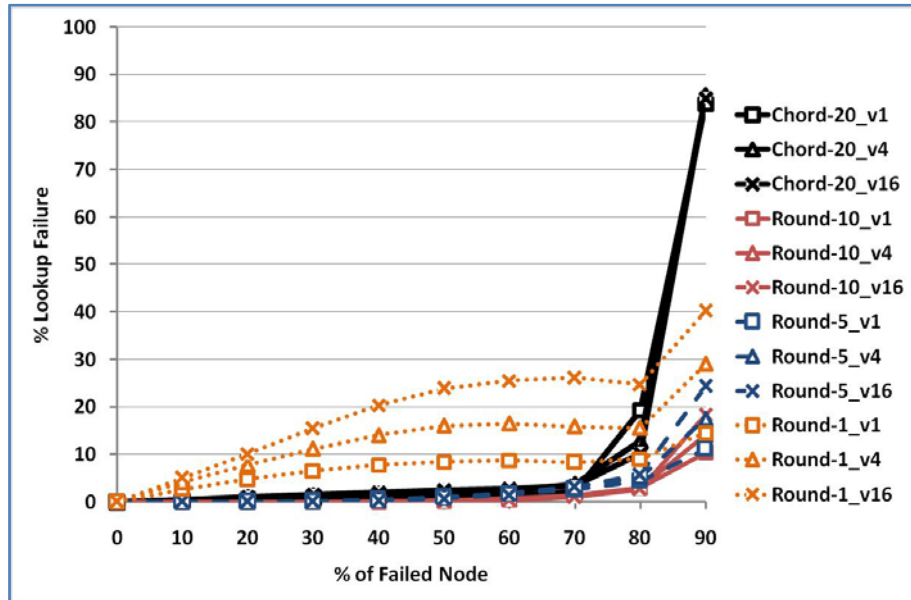


Figure 7.12: Failing Scenario - Lookup failure percentage with 10,000 queries

The important reason for the misbehaving of Chord protocol is the appearance of the isolated nodes that do not have routing information or successors. Chord protocol distributes the responsibilities of the keys among the nodes that results in lookup failure when the nodes cannot reach the responsible node and the lookup failure increases when the nodes are ISOLATED. On the other

hand, while the number of queries will not affect the performance of Chord-20, it has a large impact on round-robin configurations as evident from Figure 7.13. This is because there will be sufficient queries to remove the FAILED nodes from the routing tables during the lookup process.

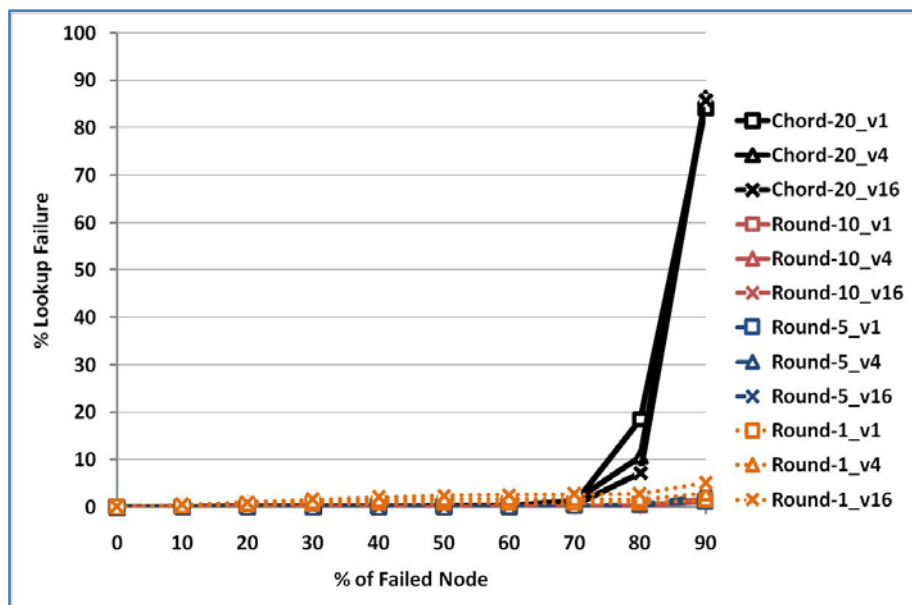


Figure 7.13: Failing Scenario - Lookup failure percentage with 100,000 queries

#### 7.3.3.2. Load Balance

Similar to the blocking scenario, the average load between the nodes follows the ideal average and the fairness has the same level as the blocking scenario. However, both the average load and the fairness drop in Chord-20 at 90% because



of the increase in the lookup failure as shown in Figure 7.14, Figure 7.15 and Figure 7.16.

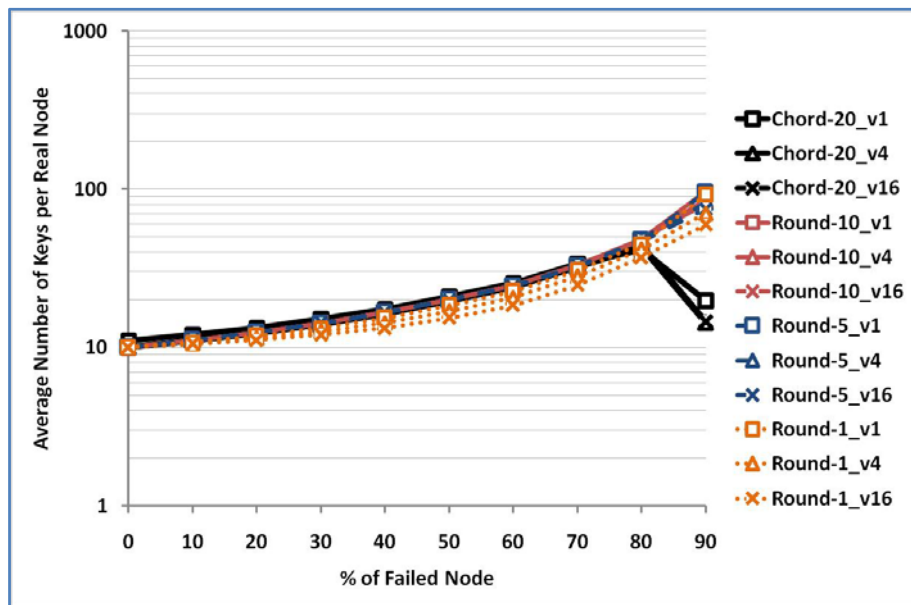


Figure 7.14: Failing Scenario - Average load between the resolvers

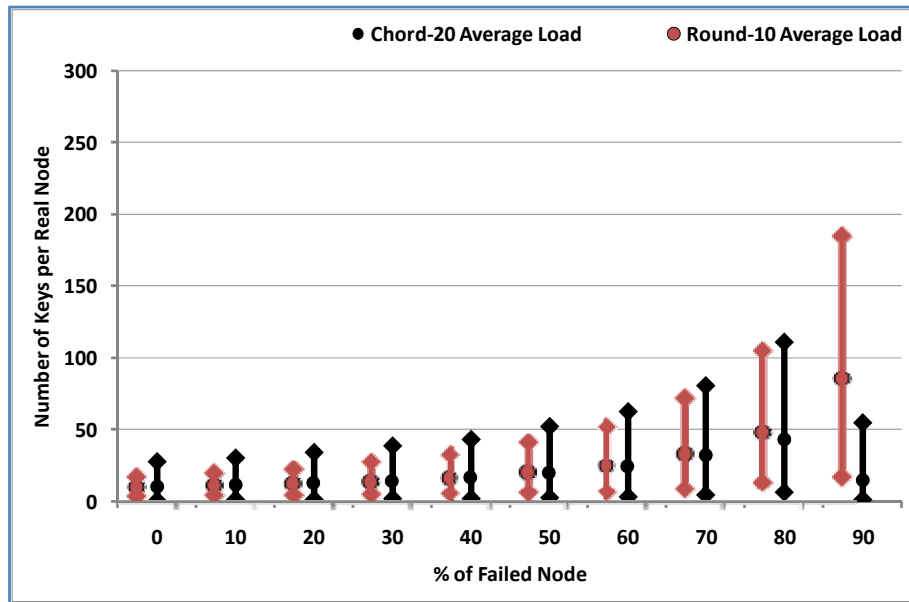


Figure 7.15: Failing Scenario – The average, 1<sup>st</sup> and 99<sup>th</sup> percentiles, load between the resolvers in Round-10 and Chord-20 with four virtual nodes

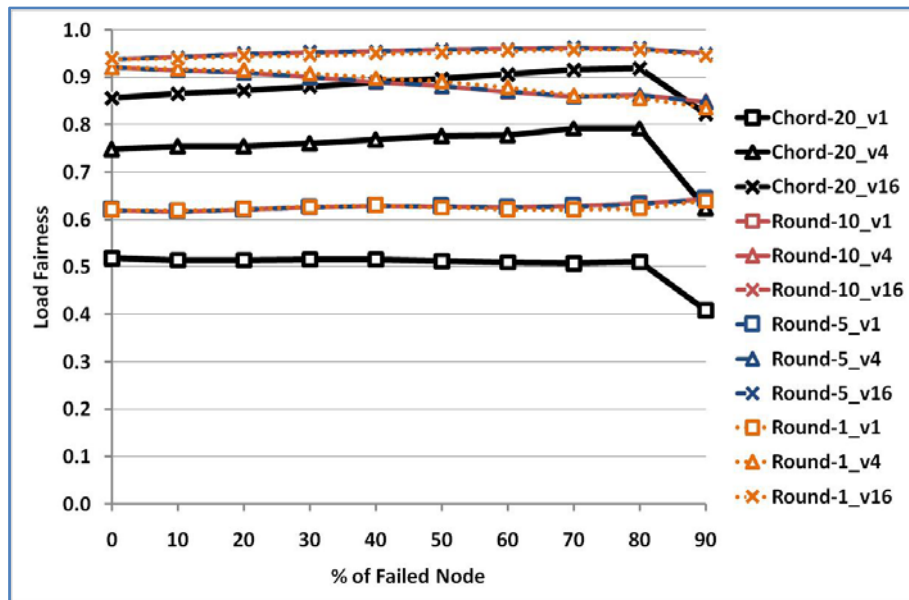


Figure 7.16: Failing Scenario - Load fairness between the resolvers

### 7.3.3.3. Message Path Length

Chord-20 experiences an increase in the path length because of the failed nodes as shown in Figure 7.17. When the network stabilizes such that there are no failed nodes in the routing tables, the path length pattern follows the equation provided by Stoica et al.[3] as evident from Figure 7.18. The differences at 90% are caused by the number of virtual nodes per real node. Contrarily to Chord-20, there are no hops in resolving queries in the round-robin configurations.

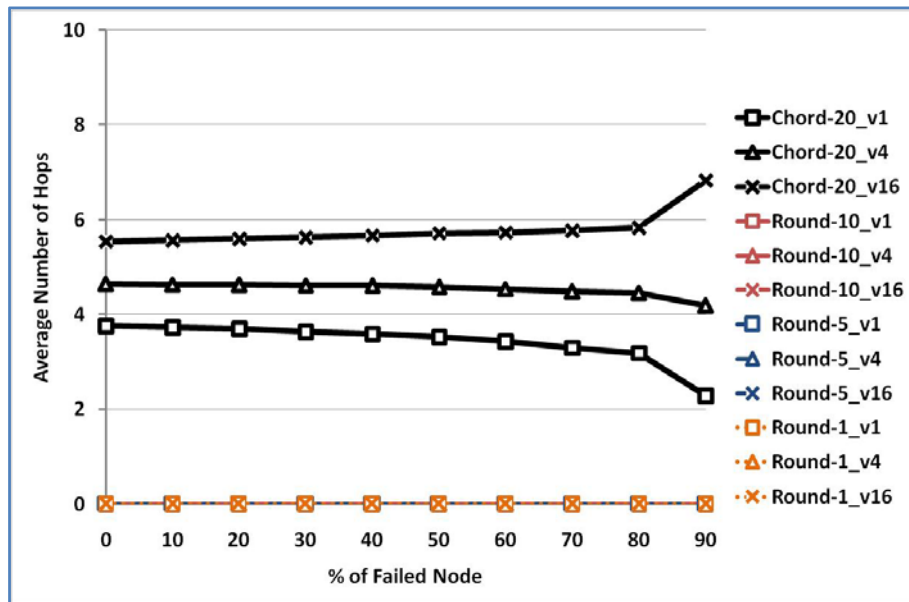


Figure 7.17: Failing Scenario - Successful lookup path length with 10,000 queries

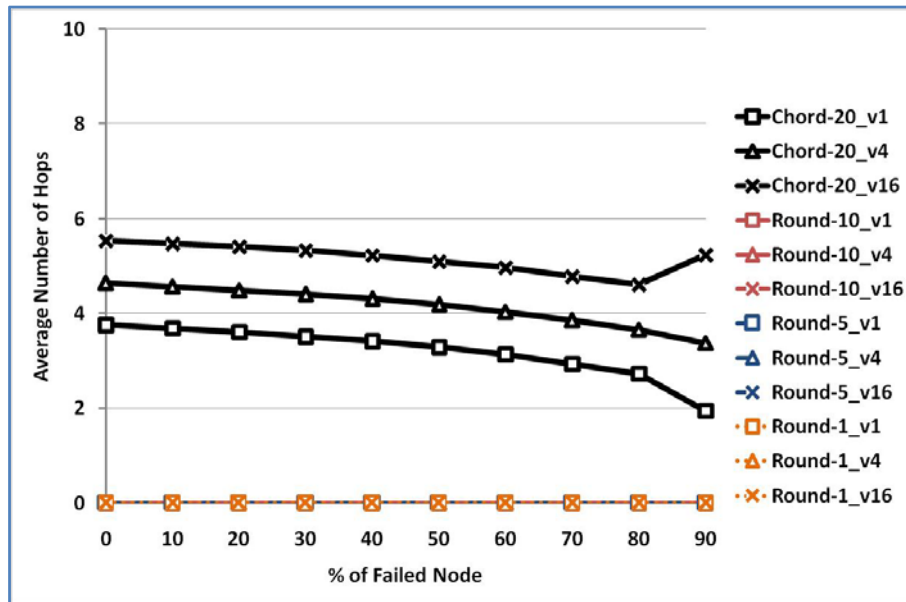


Figure 7.18: Failing Scenario - Successful lookup path length with 100,000 queries

#### 7.3.3.4. Message Timeout

Because the timeout is calculated for the successful lookups only, there is no timeout in Round-1. On the other hand, the other configurations have to check the number of nodes (i.e., some of them are DEAD) during the lookup that caused longer timeout as shown in Figure 7.19. The reason of the drop in the timeout is the lookup failure behavior as was presented in Figure 7.12. However, the timeout will be negligible after the network stabilizes (i.e., increase the number of the queries) as evident from Figure 7.20.

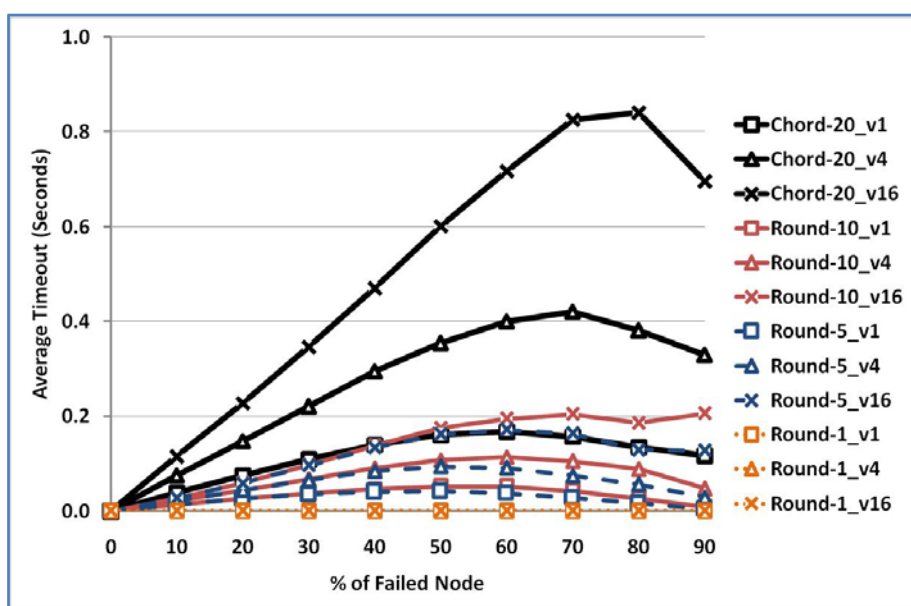


Figure 7.19: Failing Scenario - Successful lookup timeout with 10,000 queries

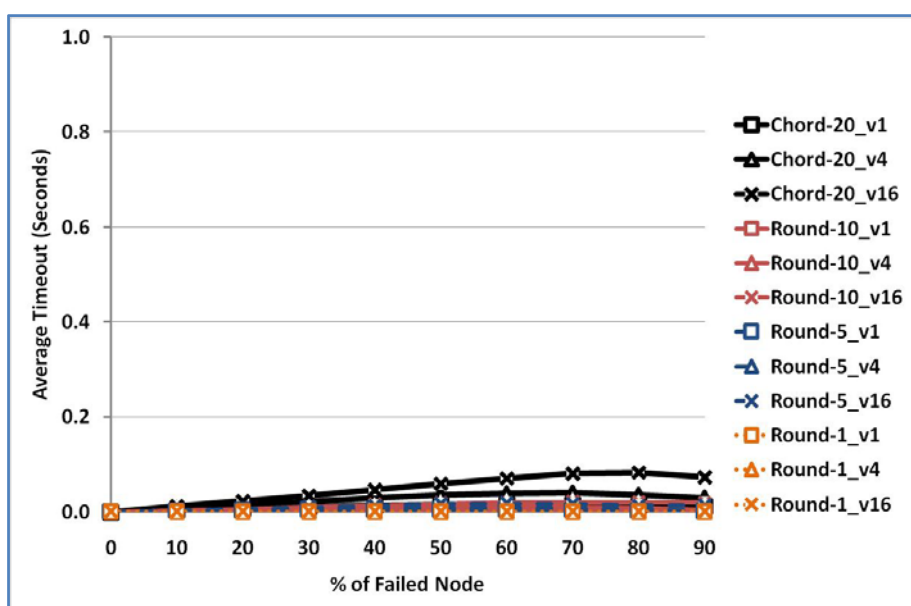


Figure 7.20: Failing Scenario - Successful lookup timeout with 100,000 queries

### 7.3.3.5. Traffic Balance

As shown in Figure 7.21, the average traffic between the nodes increased compared with the average traffic in the blocking scenario presented in Figure 7.9 for both Chord and round-robin configurations. This happens in the failing scenario because the only nodes participating in the system are the ALIVE nodes; there are no BLOCKING nodes in the failing scenario.

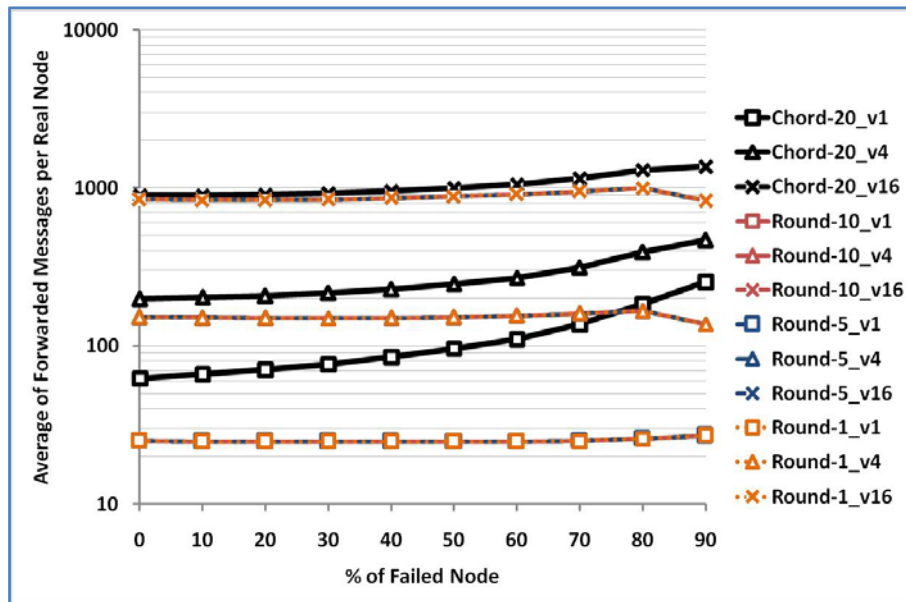


Figure 7.21: Failing Scenario - Average traffic between the nodes

On the other hand, the fairness of the traffic is stable, as depicted in Figure 7.22, until the lookup failure occurs in the behavior of Chord-20. This will affect the

stabilization process in all configurations because it only depends on the enhanced Chord; in correlation with the behavior demonstrated earlier in Figure 7.12. From that moment on the fairness of the traffic starts to drop as depicted in Figure 7.22.

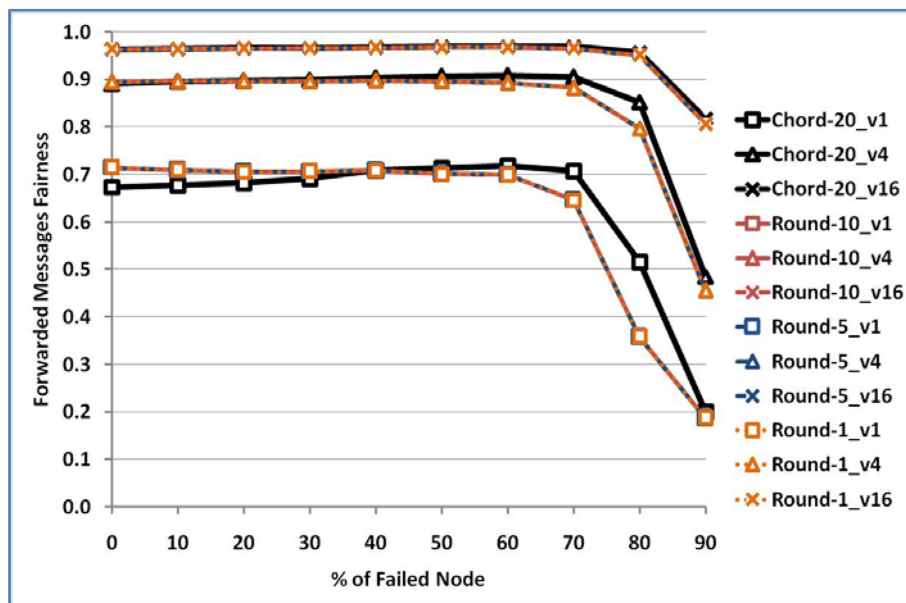


Figure 7.22: Failing Scenario - Traffic fairness between the nodes

#### 7.3.4. Observations

- There are no differences between round-robin configurations except in the message timeout metric when the network has not stabilized yet.
- Round-robin configurations could be considered failure free after the network has stabilized. In contrast, Chord-20 will continue to have lookup

failure in the high nodes failure scenario even after the network has stabilized.

- Round-robin configurations have higher fairness compared to Chord-20.
- The effect of the virtual nodes is the same as what is observed in the case of the blocking scenario.
- Round-10 with four virtual nodes has better performance than Chord-20 with four virtual nodes when the failing nodes are less than 50 %. It has no lookup failure, low traffic, and higher fairness.

## **7.4. Joining and Leaving Scenario**

### **7.4.1. Objective**

The objective of the joining and leaving scenario is to study the behavior of resolving the queries with the proposed solution when there are joining and leaving nodes in the network. The behavior is studied for both the modified Chord and the round-robin approach.



### 7.4.2. Scenario Description

After a network of 1,000 real nodes is stable, the rate of joining and leaving nodes per second in the network is varied from 0.1 to 1.0 with increments of 0.1 in each sample. During the simulation time, uniform random queries are generated using the default parameters.

This scenario combines the characteristics of the previous scenarios. It has the notifying feature as in the blocking scenario where the node will notify its neighbors before departing. Also, the departed node will be considered dead when it appears in the Successor list or in the finger table which reflects the failure scenario.

### 7.4.3. Scenario Results

#### 7.4.3.1. Lookup Failure

As previously stated, the node will notify its neighbor before departing in order to guarantee the correctness of the Successor List. Therefore, the lookup queries are always resolved in Chord-20 as evident from Figure 7.23. However, the finger table cannot be updated completely between two successive joined/departed nodes. This happens because the finger tables need at least  $m$

stabilization periods to be completely updated and the churn rate in this study is much smaller than  $m$  stabilization periods. As a result there would be an increase in the lookup failure in the round-robin configurations as observed from Figure 7.23.

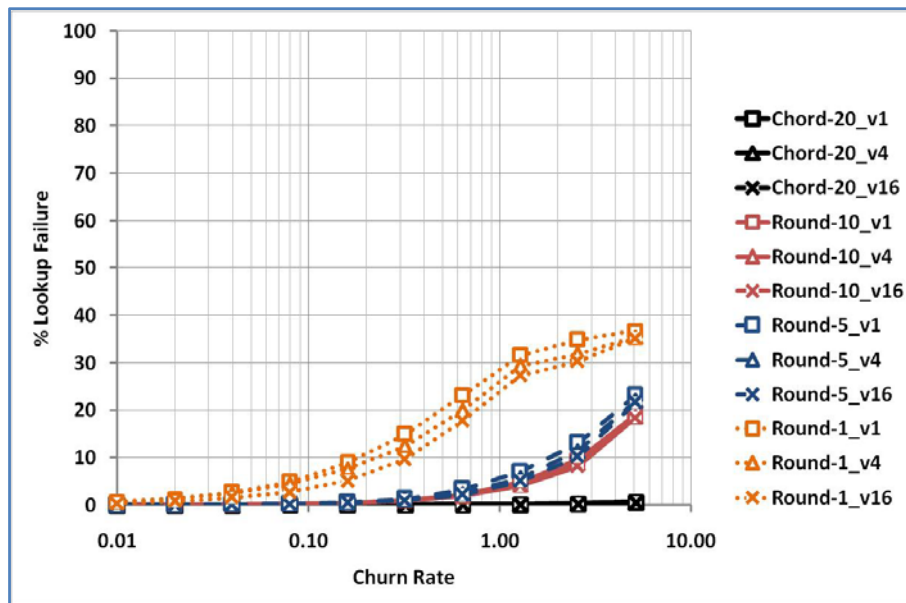


Figure 7.23: Joining & Leaving Scenario - Lookup failure percentage

#### 7.4.3.2. Load Balance

Although the average load between all nodes that joined or departed the network fits the load equation presented in section 1; as shown in Figure 7.24 and Figure 7.25, the fairness between these nodes is not equal. For example, for low

churn rate (0.01 – 0.1), the 10,000 queries will be resolved with minimum change in the routing tables. Thus, the round-robin configurations show better fairness when compared to Chord-20 as shown in Figure 7.26. This behavior is reversed in the case of large churn rate ( $\geq 1.0$ ) because of the increase in the lookup failure in the round-robin configurations as presented earlier in Figure 7.23.

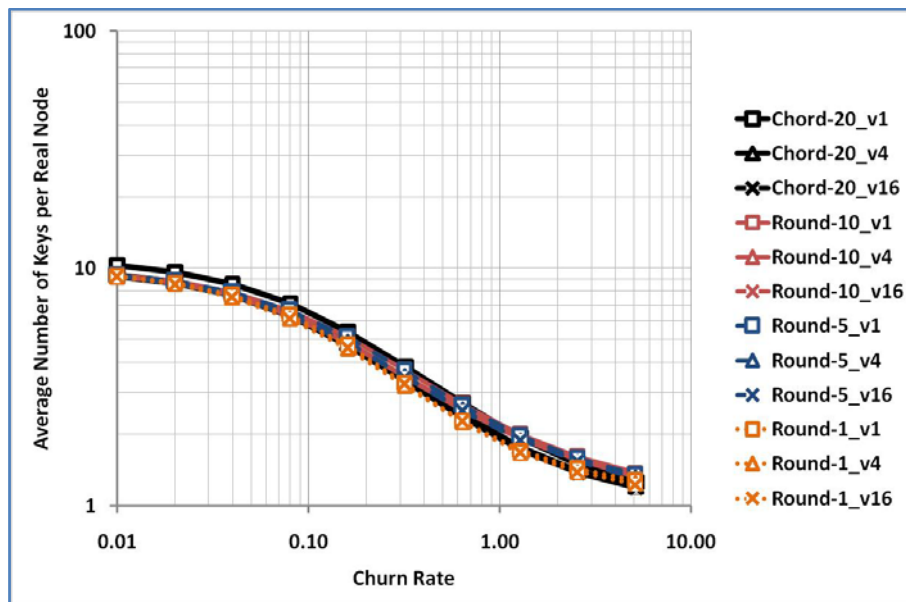


Figure 7.24: Joining & Leaving Scenario - Average load between the resolvers

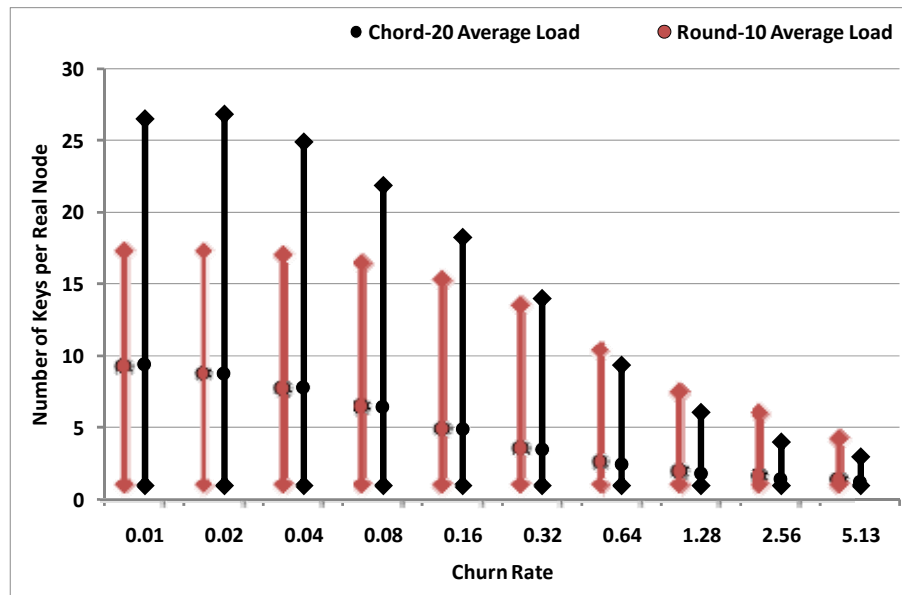


Figure 7.25: Joining & Leaving Scenario – The average, 1st and 99th percentiles, load between the resolvers in Round-10 and Chord-20 with four virtual nodes

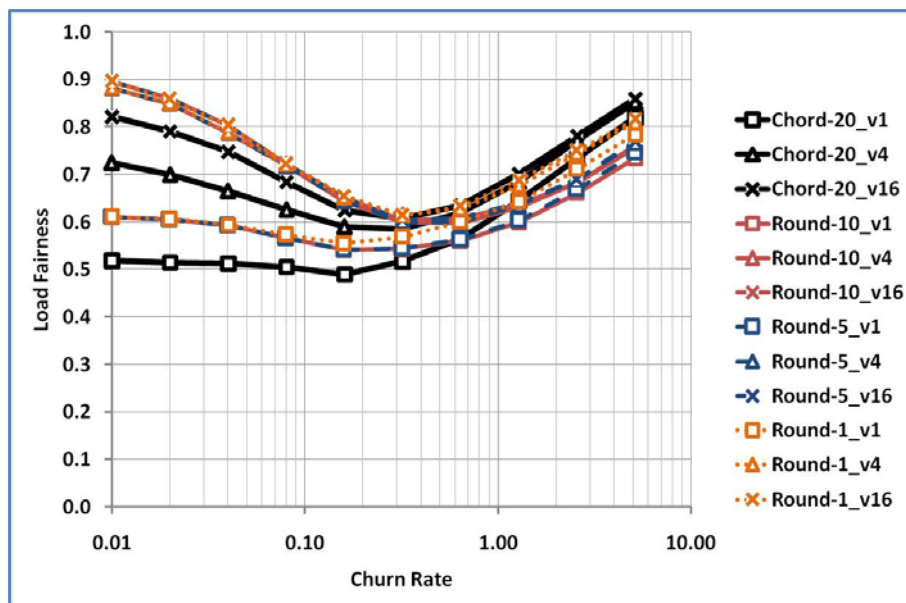


Figure 7.26: Joining & Leaving Scenario - Load fairness between the resolvers

### 7.4.3.3. Message Path Length

Because of the increase in the number of failing nodes in the finger table, the path length in Chord-20 increases proportionally to the churn rate as depicted in Figure 7.27. On the other hand, the path length in the round-robin configurations is not affected by the joining and leaving scenario.

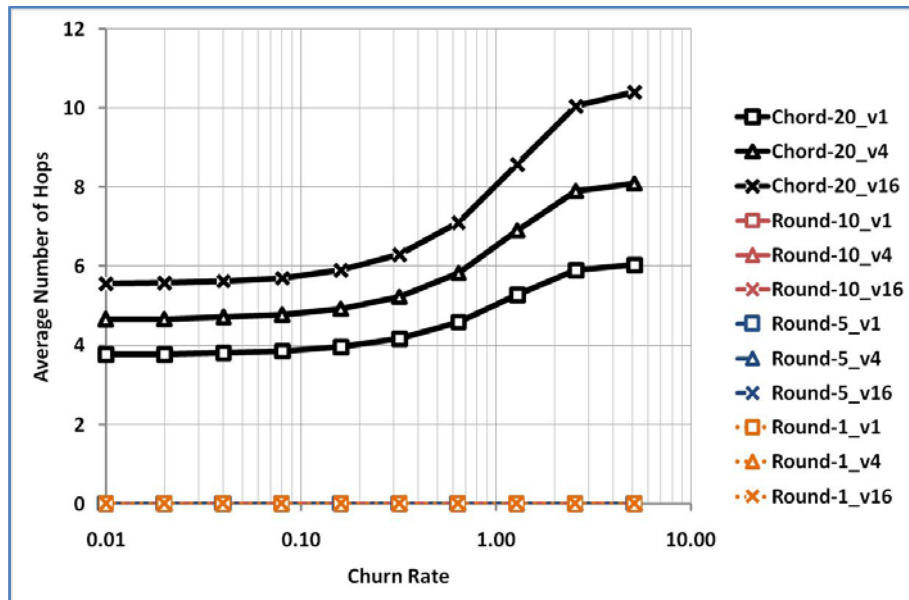


Figure 7.27: Joining & Leaving Scenario - Successful lookup path length

#### 7.4.3.4. Message Timeout

The slow update of the finger tables results in having some of the DEPARTED nodes still being presented as entries in the finger tables. Accordingly, during a lookup such finger table entries will cause an increase in the timeout as observed from Figure 7.28. Also, because Chord-20 depends on forwarding to resolve queries, it has higher timeout than the round-robin configurations.

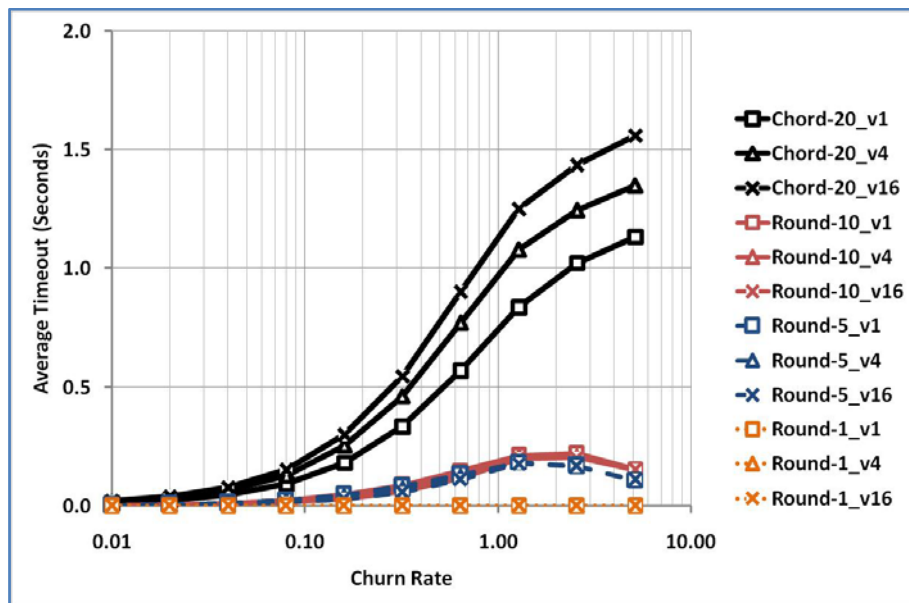


Figure 7.28: Joining & Leaving Scenario - Successful lookup timeout

#### 7.4.3.5. Traffic Balance

With small rate Churn rate, the nodes that joined the system will stay in the system for a while. Thus, the average traffic per node is higher with the small Churn rate than the higher Churn rate where the nodes in the higher Churn rate could join the system and leave without any contribution as shown in Figure 7.29. On the other hand, the fairness between nodes is inversely proportional to the Churn rate because of the increasing number of the non-contributed nodes.

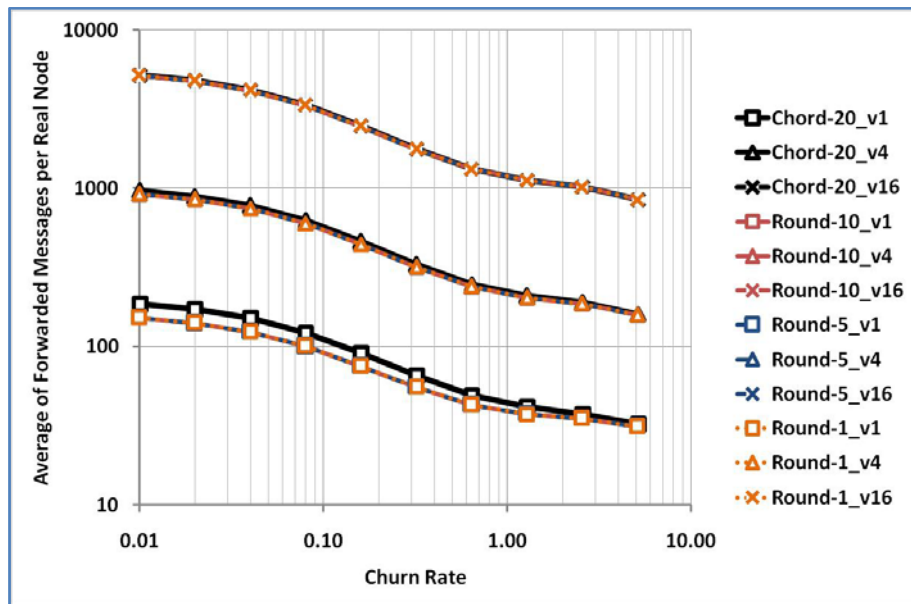


Figure 7.29: Joining & Leaving Scenario - Average traffic between the nodes

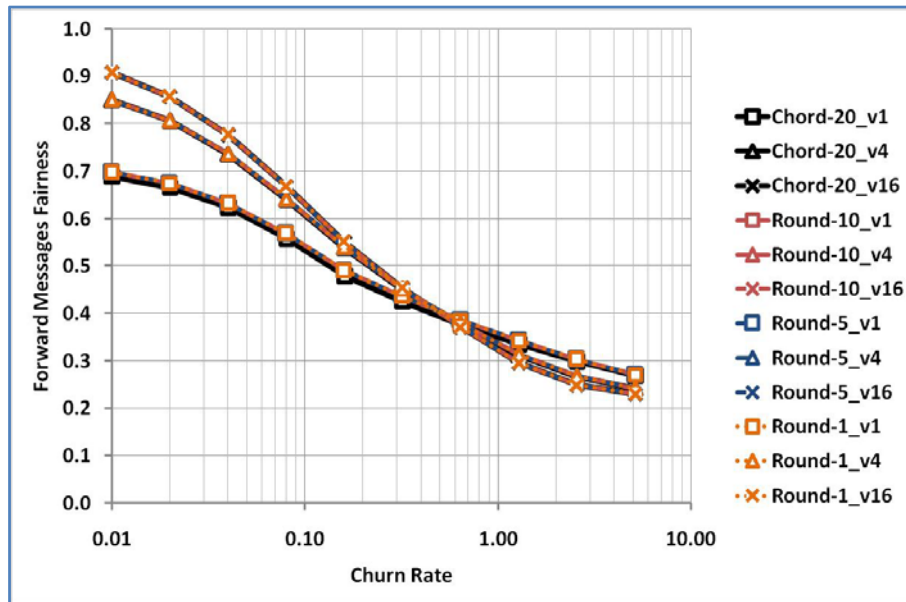


Figure 7.30: Joining & Leaving Scenario - Traffic fairness between the nodes

#### 7.4.4. Observations

- Round-robin is not suitable for high churn rate ( $\geq 0.1$ ).
- The virtual nodes have little effect on the performance of the round-robin.
- Chord-20 suffers from a high latency and path length proportional to the Churn rate.
- For low churn rate ( $\leq 0.1$ ), Round-10 with four virtual nodes has better performance than Chord-20. It has no lookup failure, low traffic, and higher fairness.



## 7.5. Evaluation and Recommendations

The three scenarios, blocking scenario, failure scenario, and joining and leaving scenario, study the performance of both the proposed round-robin solution and the modified Chord taking into account the worst case conditions. However, because the solution is proposed with respect to the DNS system, the worst case conditions are considered extreme conditions, especially when it is expected that the DNS system will have high performance servers with low churn rate. Therefore, the dynamic round-robin P2P solution is considered a good solution acting as a secondary path to resolve the lookup queries.

From the simulation results, it can be noted that Round-10 with 4 virtual nodes has almost the same behavior as Chord-20. Moreover, it has no hops, low timeout, no lookup failure, and better fairness. The 4 virtual nodes case is chosen because it has a similar performance as the 16 virtual nodes case but with less traffic. The charts in Figure 7.31 - Figure 7.37 that are extracted from earlier figures compare between Round-10 and Chord-20 assuming that 10,000 queries were submitted. Similar to the worst case scenario stated in [3], the charts compare the results of the blocking scenario and failing scenario at 50% nodes blocked or failed, respectively. In addition, the node joining and leaving with churn rate of 0.01 is included even though it is considered a fast rate when talking about a P2P system

between the DNS servers. The charts clearly demonstrate the advantage of using Round-10 over Chord-20. Hence, it is recommended to use Round-10 for a P2P system between the DNS servers.

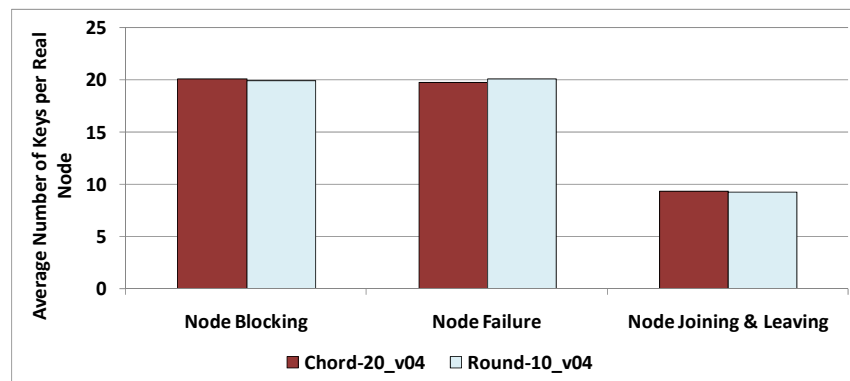


Figure 7.31: Comparison between Round-10 and Chord-20: Load

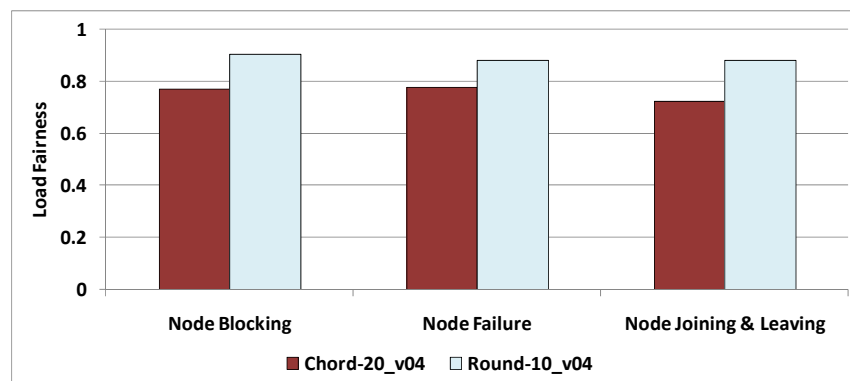


Figure 7.32: Comparison between Round-10 and Chord-20: Load Fairness

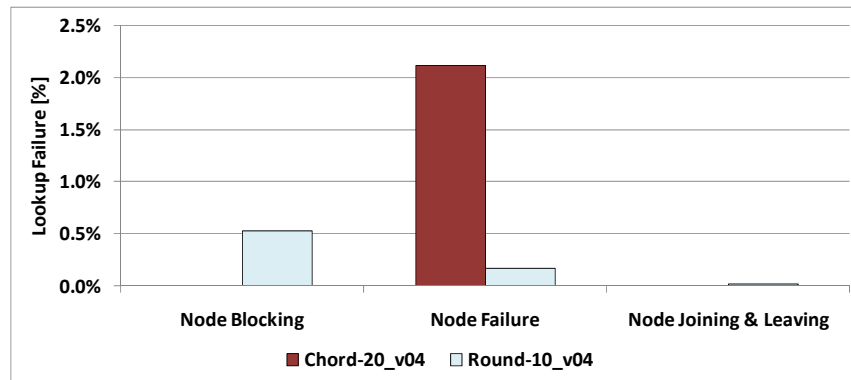


Figure 7.33: Comparison between Round-10 and Chord-20: Lookup Failure

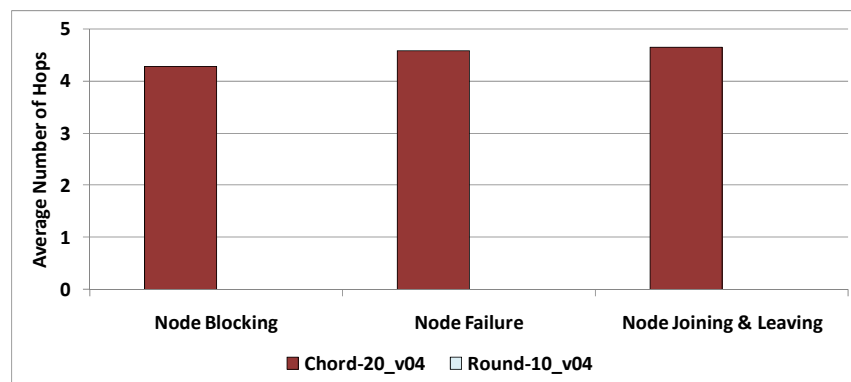


Figure 7.34: Comparison between Round-10 and Chord-20: Path length

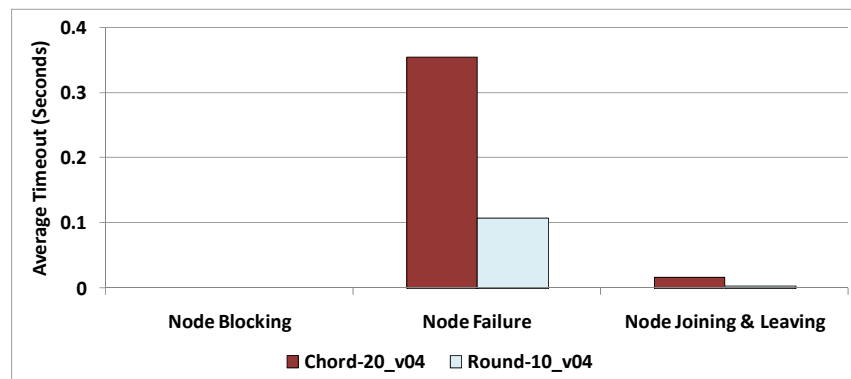


Figure 7.35: Comparison between Round-10 and Chord-20: Timeout

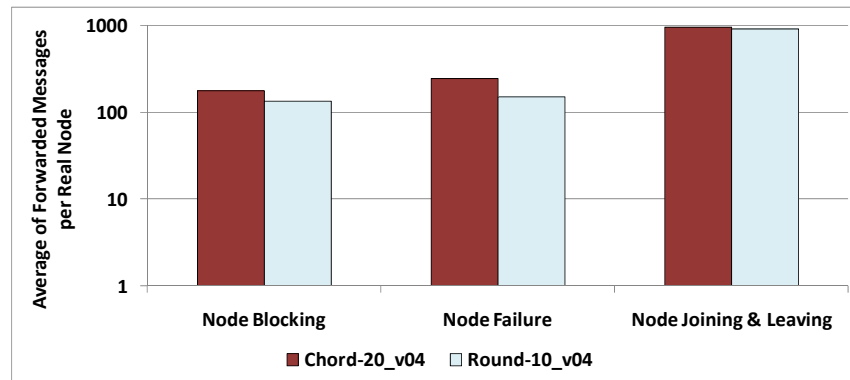


Figure 7.36: Comparison between Round-10 and Chord-20: Traffic

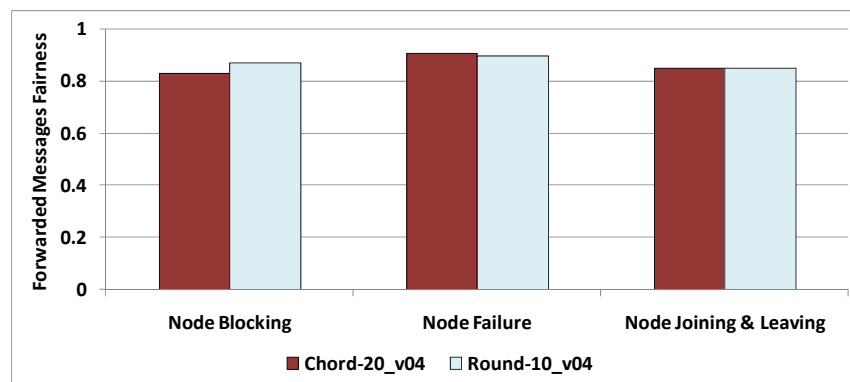


Figure 7.37: Comparison between Round-10 and Chord-20: Traffic Fairness

It should be noted that the dynamic round-robin P2P solution is dependent on the finger table of the underlying Chord P2P. Hence, it could behave better by enhancing the updating algorithm of the finger table. Also, modifying the notify neighbors process by adding some fingers to the notification would enhance the performance of the proposed solution further.

## CHAPTER 8

### CONCLUSION

Many papers proposed a replacement to the existing DNS or a mechanism to distribute the DNS space among the name servers; i.e., [27], [32], and [33]. This thesis proposes a solution that provides alternative paths to resolve the query which increases the availability of the DNS to specific regions such as KSA. To minimize the effects of the intentional blocking by a higher name server, a secondary DNS lookup service using P2P system using the local DNS resolvers was proposed. The P2P system is based on the Chord protocol and uses the round-robin scheme when communicating with other nodes in the routing table. This solution provides the load balance between the nodes in the system and provides an answer for the DNS query through two parallel paths without modifying the existing DNS system or increasing the latency, the standard path, i.e., through the root DNS or TLD DNS, and the P2P system. The following is the list of thesis contributions:

1. Study the performance of Chord protocol taking into account the worst case conditions.

2. Develop and implement a new robust solution to overcome the intentional DNS blocking from higher DNS servers without any modification to the existing DNS standard.
3. Implement an object oriented simulator to simulate the modified Chord protocol and the dynamic round-robin P2P solution.
4. Study the performance of the solution from different aspects taking into account the worst case conditions.

## **8.1. Future Work**

1. Study the performance of the solution under different P2P protocols.
2. Enhance Chord protocol by modifying the finger table update algorithm.
3. Study the impact of the solution on the resolve time when integrated with the existing DNS.
4. Enhance the performance of the solution by taking into account the geographical locality aspects.
5. Study the performance of the solution with zip-f distribution traffic.

## REFERENCES

- [1] Wikipedia. Wikipedia, the free encyclopedia. [Online].  
<http://en.wikipedia.org/wiki/Internet>
- [2] Root hints - ICANNwiki. [Online]. [http://icannwiki.org/Root\\_hints](http://icannwiki.org/Root_hints)
- [3] Ion Stoica et al., "Chord: A Scalable Peer-to-Peer Lookup Protocol for Internet Applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, February 2003.
- [4] P. Mockapetris, Domain Names - Concepts and Facilities, November 1987, RFC1034.
- [5] Domain Name System - Wikipedia, the free encyclopedia. [Online].  
[http://en.wikipedia.org/wiki/Domain\\_name\\_system](http://en.wikipedia.org/wiki/Domain_name_system)
- [6] J. Postel, Domain Name System Structure and Delegation, March 1994, RFC1591.
- [7] Bruce Greenblatt, *Internet Directories: How to Build and Manage Applications for LDAP, DNS, and Other Directories*, 1st ed.: Prentice Hall PTR, 2000.
- [8] P. Mockapetris, Domain Names - Implementation and Specification, November 1987, RFC1035.

- [9] Keong Lua, Jon Crowcroft, Marcelo Pias, and Steven Lim, "A Survey And Comparison Of Peer-To-Peer Overlay Network Schemes," *IEEE Communications Surveys & Tutorials*, vol. 7, no. 2, Second Quarter 2005.
- [10] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong, Freenet: A Distributed Anonymous Information Storage and Retrieval System, 1999, <http://freenetproject.org/freenet.pdf>.
- [11] Gnutella development forum, the Gnutella v0.6 protocol, 2001, [http://groups.yahoo.com/group/the\\_gdf/files/](http://groups.yahoo.com/group/the_gdf/files/).
- [12] Fasttrack Peer-to-Peer Technology Company, 2001, <http://www.fasttrack.nu/>.
- [13] Kazaa Media Desktop, 2001, <http://www.kazaa.com/>.
- [14] Bittorrent, 2003, <http://bitconjurer.org/BitTorrent/>.
- [15] The Overnet File-sharing Network, 2002, <http://www.overnet.com>.
- [16] Overnet/edonkey2000, 2000, <http://www.edonkey2000.com>.
- [17] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker, "A Scalable Content Addressable Network," *Proc. ACM SIGCOMM*, p. 161-72, 2001.
- [18] B.Y. Zhao et al., "Tapestry: A Resilient Global-Scale Overlay for Service Deployment," *IEEE JSAC*, vol. 22, no. 1, pp. 41-53, Jan. 2004.



- [19] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan, "Chord A Scalable Peer-to-peer Lookup Service for Internet Applications," *ACM SIGCOMM*, vol. 31, no. 4, pp. 149 - 160, Oct. 2001.
- [20] Antony I. T. Rowstron and Peter Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, 2001, pp. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems.
- [21] P. Maymounkov and D. Mazieres, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," *Proc. IPTPS Cambridge*, p. 53-65, Feb. 2002.
- [22] D. Malkhi, M. Naor, and D. Ratajczak, "Viceroy: A Scalable and Dynamic Emulation of the Butterfly," *Proc. ACM PODC*, p. 183-92, July 2002.
- [23] D. Barr, Common DNS Operational and Configuration Errors, February 1996, RFC1912.
- [24] Vasileios Pappas, Dan Massey, and Lixia Zhang, "Enhancing DNS Resilience against Denial of Service Attacks," in *the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 2007, pp. 450-459.
- [25] George Lawton, "Stronger Domain Name System Thwarts Root- Server Attacks," *Computer*, vol. 40, no. 5, pp. 14-17, May 2007.

- [26] Steven Cheung, "Denial of Service against the Domain Name System: Threats and Countermeasures," *IEEE Security and Privacy*, vol. 4, no. 1, p. 40, January 2006.
- [27] Nayot Poolsappasit and Indrajit Ray, "Enhancing Internet Domain Name System Availability by Building Rings of Cooperation Among Cache Resolvers," in *Workshop on Information Assurance*, United States Military Academy, West Point, NY, June 2007.
- [28] Suranjith Ariyapperuma and Chris J. Mitchell, "Security Vulnerabilities in DNS and DNSSEC," in *Proceedings of the The Second International Conference on Availability, Reliability and Security*, 2007, pp. 335-342.
- [29] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose., DNS security introduction and requirements, Mar. 2005, RFC 4033.
- [30] Hitesh Ballani and Paul Francis, "Mitigating DNS DoS Attacks," in *the 15th Conference on Computer and Communications Security*, Alexandria, Virginia, USA, 2008, pp. 189-198.
- [31] Edith Cohen and Haim Kaplan, Proactive Caching of DNS Records: Addressing a Performance Bottleneck, April 22, 2003.

- [32] Venugopalan Ramasubramanian and Emin Gün Sirer, "The Design and Implementation of a Next Generation Name Service for the Internet," in *Applications, technologies, architectures, and protocols for computer communications*, Portland, Oregon, USA, 2004, pp. 331 - 342.
- [33] Russ Cox, Athicha Muthitacharoen, and Robert Morris, "Serving DNS Using a Peer-to-Peer Lookup Service," in *First International Workshop on Peer-to-Peer Systems*, London, UK, 2002, pp. 155 - 165.
- [34] David Karger et al., "Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web," in *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, El Paso, Texas, United States, 1997, pp. 654 - 663.
- [35] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica, "Load Balancing in Structured P2P Systems," in *Second International Workshop on Peer-to-Peer Systems*, 2003.
- [36] Silvia Bianchi, Sabina Serbu, Pascal Felber, and Peter Kropf, "Adaptive Load Balancing for DHT Lookups," in *Computer Communications and Networks, 2006. ICCCN 2006.*, Neuchatel, Oct. 2006, pp. 411-418.
- [37] R. Bush, D. Karrenberg, M. Kisters, and R. Plzak, Root Name Server Operational Requirements, June 2000, RFC2870.
- [38] Mark Baker and Rahim Lakhoo, Peer-to-Peer Simulators, May 2007.

- [39] The Network Simulator - ns-2. [Online]. <http://www.isi.edu/nsnam/ns/>
- [40] PeerSim: A Peer-to-Peer Simulator. [Online]. <http://peersim.sourceforge.net/>
- [41] N. Kotilainen, M. Vapa, T. Keltanen, A. Auvinen, and J. Vuori, "P2PRealm - peer-to-peer network simulator," pp. 93 - 99, July 2006.
- [42] S. Naicken et al., "The state of peer-to-peer simulators and simulations," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 2, pp. 95 - 98, April 2007.

## **VITA**

- Name: FAHD AHMED ABDULRAHMAN ABDULHAMEED
- Nationality: Yemeni
- Present address: P.O.Box 963, KFUPM, Dhahran 31261, Saudi Arabia
- Permanent address: Al-Baghdadiyah Al-Sharqiya, Jeddah, Saudi Arabia
- Email address: fahd.abdulhamid@gmail.com
- Phone number: +966506825969
- Bachelor of Science in Computer Engineering from KFUPM - 2003
- Master of Science in Computer Networks from KFUPM - 2011